Numerical Methods

of

Exploration Seismology

with algorithms in MATLAB

Gary F. Margrave

Department of Geology and Geophysics

The University of Calgary

July 11, 2003

Preface

The most important thing to know about this draft is that it is unfinished. This means that it must be expected to have rough passages, incomplete sections, missing references, and references to nonexistent material. Despite these shortcomings, I hope this document will be welcomed for its description of the CREWES MATLAB software and its discussion of velocity, raytracing and migration algorithms. I only ask that the reader be tolerant of the rough state of these notes. Suggestions for the improvement of the present material or for the inclusion of other subjects are most welcome.

Exploration seismology is a complex technology that blends advanced physics, mathematics and computation. Often, the computational aspect is neglected in teaching because, traditionally, seismic processing software is part of an expensive and complex system. In contrast, this book is structured around a set of computer algorithms that run effectively on a small personal computer. These algorithms are written in MATLAB code and so the reader should find access to a MATLAB installation. This is not the roadblock that it might seem because MATLAB is rapidly gaining popularity and the student version costs little more than a typical textbook.

The algorithms are grouped into a small number of *toolboxes* that effectively extend the functionality of MATLAB. This allows the student to experiment with the algorithms as part of the process of study. For those who only wish to gain a conceptual overview of the subject, this may not be an advantage and they should probably seek a more appropriate book. On the other hand, those who wish to master the subject and perhaps extend it through the development of new methods, are my intended audience. The study of these algorithms, including running them on actual data, greatly enriches learning.

The writing of this book has been on my mind for many years though it has only become physical relatively recently. The material is the accumulation of many years of experience in both the hydrocarbon exploration industry and the academic world. The subject matter encompasses the breadth of exploration seismology but, in detail, reflects my personal biases. In this preliminary edition, the subject matter includes raytracing, elementary migration, some aspects of wave-equation modelling, and velocity manipulation. Eventually, it will also include theoretical seismograms, wavelets, amplitude adjustment, deconvolution, filtering (1D and 2D), statics adjustment, normal moveout removal, stacking and more. Most of the codes for these purposes already exists and have been used in research and teaching at the University of Calgary since 1995.

During the past year, Larry Lines and Sven Treitel have been a constant source of encouragement and assistance. Pat Daley's guidance with the raytracing algorithms has been most helpful. Pat, Larry, Sven, Dave Henley, and Peter Manning have graciously served as editors and test readers. Many students and staff with CREWES have stress-tested the MATLAB codes. Henry Bland's technical computer support has been essential. Rob Stewart and Don Lawton have been there with moral support on many occasions.

I thank you for taking the time to examine this document and I hope that you find it rewarding.

Contents

Chapter 1

Introduction

1.1 Scope and Prerequisites

This is a book about a complex and diverse subject: the numerical algorithms used to process exploration seismic data to produce images of the earth's crust. The techniques involved range from simple and graphical to complex and numerical. More often than not, they tend towards the latter. The methods frequently use advanced concepts from physics, mathematics, numerical analysis, and computation. This requires the reader to have a background in these subjects at approximately the level of an advanced undergraduate or beginning graduate student in geophysics or physics. This need not include experience in exploration seismology but such would be helpful.

Seismic datasets are often very large and have historically strained computer storage capacities. This, along with the complexity of the underlying physics, has also strongly challenged computation throughput. These difficulties have been a significant stimulus to the development of computing technology. In 1980, a 3D migration[1] was only possible in the advanced computing centers of the largest oil companies. At that time, a 50,000 trace 3D dataset would take weeks to migrate on a dedicated, multi-million dollar, computer system. Today, much larger datasets are routinely migrated by companies and individuals around the world, often on computers costing less than $5000. The effective use of this book, including working the computer exercises, requires access to a significant machine (at least a late-model PC or Macintosh) with MATLAB installed and having more than 64 mb of RAM and 10 gb of disk.

Though numerical algorithms, coded in MATLAB, will be found throughout this book, this is not a book primarily about MATLAB. It is quite feasible for the reader to plan to learn MATLAB concurrently with working through this book, but a separate reference work on MATLAB is highly recommended. In addition to the reference works published by The MathWorks (the makers of MATLAB), there are many excellent, independent guides in print such as Etter (1996), Hanselman and Littlefield (1998), Redfern and Campbell (1998) and the very recent Higham and Higham (2000). In addition, the student edition of MATLAB is a bargain and comes with a very good reference manual. If you already own a MATLAB reference, then stick with it until it proves inadequate. The website of The MathWorks is worth a visit because it contains an extensive database of books about MATLAB.

Though this book does not teach MATLAB at an introductory level, it illustrates a variety of advanced techniques designed to maximize the efficiency of working with large datasets. As

[1] *Migration* refers to the fundamental step in creating an earth image from scattered data.

with many MATLAB applications, it helps greatly if the reader has had some experience with linear algebra. Hopefully, the concepts of matrix, row vector, column vector, and systems of linear equations will be familiar.

1.1.1 Why MATLAB?

A few remarks are appropriate concerning the choice of the MATLAB language as a vehicle for presenting numerical algorithms. Why not choose a more traditional language like C or Fortran, or an object oriented language like C++ or Java?

MATLAB was not available until the latter part of the 1980's, and prior to that, Fortran was the language of choice for scientific computations. Though C was also a possibility, its lack of a built-in facility for complex numbers was a considerable drawback. On the other hand, Fortran lacked some of C's advantages such as structures, pointers, and dynamic memory allocation.

The appearance of MATLAB changed the face of scientific computing for many practitioners, this author included. MATLAB evolved from the Linpack package that was familiar to Fortran programmers as a robust collection of tools for linear algebra. However, MATLAB also introduced a new vector-oriented programming language, an interactive environment, and built-in graphics. These features offered sufficient advantages that users found their productivity was significantly increased over the more traditional environments. Since then, MATLAB has evolved to have a large array of numerical tools, both commercial and shareware, excellent 2D and 3D graphics, object-oriented extensions, and a built-in interactive debugger.

Of course, C and Fortran have evolved as well. C has led to C++ and Fortran to Fortran90. Though both of these languages have their adherents, neither seems to offer as complete a package as does MATLAB. For example, the inclusion of a graphics facility in the language itself is a major boon. It means that MATLAB programs that use graphics are standard throughout the world and run the same on all supported platforms. It also leads to the ability to graphically display data arrays at a breakpoint in the debugger. These are useful practical advantages especially when working with large datasets.

The vector syntax of MATLAB once mastered, leads to more concise code than most other languages. Setting one matrix equal to the transpose of another through a statement like `A=B';` is much more transparent than something like

```
do i=1,n
    do j=1,m
        A(i,j)=B(j,i)
    enddo
enddo
```

Also, for the beginner, it is actually easier to learn the vector approach that does not require so many explicit loops. For someone well-versed in Fortran, it can be difficult to un-learn this habit, but it is well worth the effort.

It is often argued that C and Fortran are more efficient than MATLAB and therefore more suitable for computationally intensive tasks. However, this view misses the big picture. What really matters is the efficiency of the entire scientific process, from the genesis of the idea, through its rough implementation and testing, to its final polished form. Arguably, MATLAB is much more efficient for this entire process. The built-in graphics, interactive environment, large tool set, and strict run-time error checking lead to very rapid prototyping of new algorithms. Even in the more narrow view, well-written MATLAB code can approach the efficiency of C and Fortran. This is partly because of the vector language but also because most of MATLAB's number crunching actually happens in compiled library routines written in C.

Traditional languages like C and Fortran originated in an era when computers were room-sized behemoths and resources were scarce. As a result, these languages are oriented towards simplifying the task of the computer at the expense of the human programmer. Their cryptic syntax leads to efficiencies in memory allocation and computation speed that were essential at the time. However, times have changed and computers are relatively plentiful, powerful, and cheap. It now makes sense to shift more of the burden to the computer to free the human to work at a higher level. Spending an extra $100 to buy more RAM may be more sensible than developing a complex data handling scheme to fit a problem into less space. In this sense, MATLAB is a higher-level language that frees the programmer from technical details to allow time to concentrate on the real problem.

Of course, there are always people who see these choices differently. Those in disagreement with the reasons cited here for MATLAB can perhaps take some comfort in the fact that MATLAB syntax is fairly similar to C or Fortran and translation is not difficult. Also, The MathWorks markets a MATLAB "compiler" that emits C code that can be run through a C compiler.

1.1.2 Legal matters

It will not take long to discover that the MATLAB source code files supplied with this book all have a legal contract within them. Please take time to read it at least once. The essence of the contract is that you are free to use this code for non-profit education or research but otherwise must purchase a commercial license from the author. Explicitly, if you are a student at a University (or other school) or work for a University or other non-profit organization, then don't worry just have fun. If you work for a commercial company and wish to use this code in any work that has the potential of profit, then you must purchase a license. If you are employed by a commercial company, then you may use this code on your own time for self-education, but any use of it on company time requires a license. If your company is a corporate sponsor of CREWES (The Consortium for Research in Elastic Wave Exploration Seismology at the University of Calgary) then you automatically have a license. Under no circumstances may you resell this code or any part of it. By using the code, you are agreeing to these terms.

1.2 MATLAB conventions used in this book

There are literally hundreds of MATLAB *functions* that accompany this book (and hundreds more that come with MATLAB). Since this is not a book about MATLAB, most of these functions will not be examined in any detail. However, all have full online documentation, and their code is liberally sprinkled with comments. It is hoped that this book will provide the foundation necessary to enable the user to use and understand all of these commands at whatever level necessary.

Typographic style variations are employed here to convey additional information about MATLAB functions. A function name presented like `plot` refers to a MATLAB function supplied by The MathWorks as part of the standard MATLAB package. A function name presented like ***dbspec*** refers to a function provided with this book. Moreover, the name ***NumMethToolbox*** refers to the entire collection of software provided in this book.

MATLAB code will be presented in small numbered packages titled *Code Snippets*. An example is the code required to convert an amplitude spectrum from linear to decibel scale:

Code Snippet 1.2.1. *This code computes a wavelet and its amplitude spectrum on both linear and decibel scales. It makes Figure 1.1.*

```
1    [wavem,t]=wavemin(.002,20,.2);
2    [Wavem,f]=fftrl(wavem,t);
```

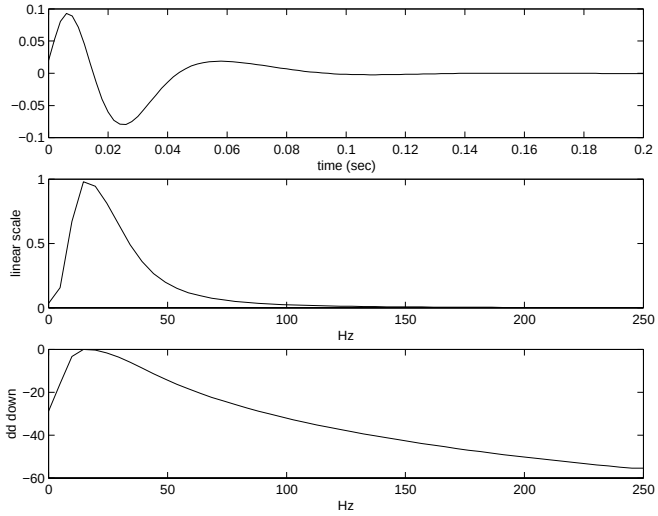

Figure 1.1: A minimum phase wavelet (top), its amplitude spectrum plotted on a linear scale (middle), and its amplitude spectrum plotted with a decibel scale (bottom).

```
3    Amp=abs(Wavem);
4    dbAmp=20*log10(Amp/max(Amp));
5    subplot(3,1,1);plot(t,wavem);xlabel('time (sec)')
6    subplot(3,1,2);plot(f,abs(Amp));xlabel('Hz');ylabel('linear scale')
7    subplot(3,1,3);plot(f,dbAmp);xlabel('Hz');ylabel('db down')
```

End Code

The actual MATLAB code is displayed in an upright typewriter font while introductory remarks are *emphasized like this*. The Code Snippet does not employ typographic variations to indicate which functions are contained in the ***NumMethToolbox*** as is done in the text proper.

It has proven impractical to discuss all of the input parameters for all of the programs shown in Code Snippets. Those germane to the current topic are discussed, but the remainder are left for the reader to explore using MATLAB 's interactive help facility. For example, in Code Snippet 1.1 ***wavemin*** creates a minimum phase wavelet sampled at .002 seconds, with a dominant frequency of 20 Hz, and a length of .2 seconds. Then ***fftrl*** computes the Fourier spectrum of the wavelet and `abs` constructs the amplitude spectrum from the complex Fourier spectrum. Finally, the amplitude spectrum is converted to decibels (db) using the formula

$$Amp_{decibels}(f) = 20 \log \left(\frac{Amp(f)}{\max(Amp(f))} \right). \tag{1.1}$$

To find out more about any of these functions and their inputs and outputs, type, for example, "help fftrl" at the MATLAB prompt.

This example illustrates several additional conventions. Seismic traces[2] are discrete time series but the textbook convention of assuming a sample interval of unity in arbitrary units is not appropriate for practical problems. Instead, two vectors will be used to describe a trace, one to give its amplitude and the other to give the temporal coordinates for the first. Thus ***wavemin*** returns two vectors (that they are vectors is not obvious from the Code Snippet) with `wavem` being the wavelet amplitudes and `t` being the time coordinate vector for `wavem`. Thus the upper part of Figure 1.1 is created by simply cross plotting these two vectors: `plot(t,wavem)`. Similarly, the Fourier spectrum, `Wavem`, has a frequency coordinate vector `f`. Temporal values must always be specified in seconds and will be returned in seconds and frequency values must always be in Hz. (One Hz is one cycle per second). Milliseconds or radians/second specifications will never occur in code though both cyclical frequency f and angular frequency ω may appear in formulae ($\omega = 2\pi f$).

Seismic traces will always be represented by column vectors whether in the time or Fourier domains. This allows an easy transition to 2-D trace gathers, such as source records and stacked sections, where each trace occupies one column of a matrix. This column vector preference for signals can lead to a class of simple MATLAB errors for the unwary user. For example, suppose a seismic trace `s` is to be windowed to emphasize its behavior in one zone and de-emphasize it elsewhere. The simplest way to do this is to create a window vector *win* that is the same length as `s` and use MATLAB's .* (dot-star) operator to multiply each sample on the trace by the corresponding sample of the window. The temptation is to write a code something like this:

Code Snippet 1.2.2. *This code creates a synthetic seismogram using the* convolutional model *but then generates an error while trying to apply a (triangular) window.*

```
1    [r,t]=reflec(1,.002,.2); %make a synthetic reflectivity
2    [w,tw]=wavemin(.002,20,.2); %make a wavelet
3    s=convm(r,w); % make a synthetic seismic trace
4    n2=round(length(s)/2);
5    win=[linspace(0,1,n2) linspace(1,0,length(s)-n2)]; % a triangular window
6    swin=s.*win; % apply the window to the trace
```

End Code

MATLAB's response to this code is the error message:

```
??? Error using ==> .* Matrix dimensions must agree.
 On line 6  ==> swin=s.*win;
```

The error occurs because the `linspace` function generates row vectors and so `win` is of size 1x501 while `s` is 501x1. The .* operator requires both operands to be have exactly the same geometry. The simplest fix is to write `swin=s.*win(:);` which exploits the MATLAB feature that `a(:)` is reorganized into a column vector (see page 26 for a discussion) regardless of the actual size of `a` .

As mentioned previously, two dimensional seismic data gathers will be stored in ordinary matrices. Each column is a single trace, and so each row is a *time slice*. A complete specification of such a gather requires both a time coordinate vector, `t`, and a space coordinate vector, `x`.

Rarely will the entire code from a function such as ***wavemin*** or ***reflec*** be presented. This is because the code listings of these functions can span many pages and contain much material that is outside the scope of this book. For example, there are often many lines of code that check input parameters and assign defaults. These tasks have nothing to do with numerical algorithms and so will not be presented or discussed. Of course, the reader is always free to examine the entire codes at leisure.

[2] A *seismic trace* is the recording of a single component geophone or hydrophone. For a multicomponent geophone, it is the recording of one component.

1.3 Dynamic range and seismic data display

Seismic data tends to be a challenge to computer graphics as well as to computer capacity. A single seismic record can have a tremendous amplitude range. In speaking of this, the term *dynamic range* is used which refers to a span of real numbers. The range of numerical voltages that a seismic recording system can faithfully handle is called its dynamic range. For example, current digital recording systems use fixed instrument gain and represent amplitudes as a 24 bit integer computer word[3]. The first bit is used to record sign while the last bit tends to fluctuate randomly so effectively 22 bits are available. This means that the amplitude range can be as large as $2^{22} \approx 10^{6.6}$. Using the definition of a decibel given in equation (1.1), this corresponds to about 132db, an enormous spread of possible values. This 132db range is usually never fully realized when recording signals for a variety of reasons, the most important being the ambient noise levels at the recording site and the instrument gain settings.

In a fixed-gain system, the instrument gain settings are determined to minimize *clipping* by the analog-to-digital converter while still preserving the smallest possible signal amplitudes. This usually means that some clipping will occur on events nearest the seismic source. A very strong signal should saturate 22-23 bits while a weak signal may only effect the lowest several bits. Thus the *precision*, which refers to the number of significant digits used to represent a floating point number, steadily declines from the largest to smallest number in the dynamic range.

1.3.1 Single trace plotting and dynamic range

Figure 1.2 was produced with Code Snippet 1.3.1 and shows two real seismic traces recorded in 1997 by CREWES [4]. This type of plot is called a *wiggle trace display*. The upper trace, called `tracefar`, was recorded on the vertical component of a three component geophone placed about 1000 m from the surface location of a dynamite shot. The lower trace, called `tracenear`, was similar except that it was recorded only 10 m from the shot. (Both traces come from the shot record shown in Figures 1.12 and 1.13.) The dynamite explosion was produced with 4 kg of explosives placed at 18 m depth. Such an explosive charge is about 2 m long so the distance from the top of the charge to the closest geophone was about $\sqrt{16^2 + 10^2} \approx 19$ m while to the farthest geophone was about $\sqrt{16^2 + 1000^2} \approx 1000$ m. The vertical axes for the two traces indicate the very large amplitude difference between them. If they were plotted on the same axes, `tracefar` would appear as a flat horizontal line next to tracenear.

Figure 1.3 (also produced with Code Snippet 1.3.1) shows a more definitive comparison of the amplitudes of the two traces. Here the *trace envelopes* are compared using a decibel scale. A trace envelope is a mathematical estimate of a bounding curve for the signal (this will be investigated more fully later in this book) and is computed using `hilbert`, that computes the complex, analytic trace ((Taner et al., 1979) and (Cohen, 1995)), and then takes the absolute value. The conversion to decibels is done with the convenience function *`todb`* (for which there is an inverse *`fromdb`*). Function *`todb`* implements equation (1.1) for both real and complex signals. (In the latter case, *`todb`* returns a complex signal whose real part is the amplitude in decibels and whose imaginary part is the phase.) By default, the maximum amplitude for the decibel scale is the maximum absolute value of the signal; but, this may also be specified as the second input to *`todb`*. Function *`fromdb`* reconstructs the original signal given the output of *`todb`*.

[3]Previous systems used a 16 bit word and variable gain. A four-bit gain word, an 11 bit mantissa, and a sign bit determined the recorded value.

[4]CREWES is an acronym for *Consortium for Research in Elastic Wave Exploration Seismology* at the University of Calgary.

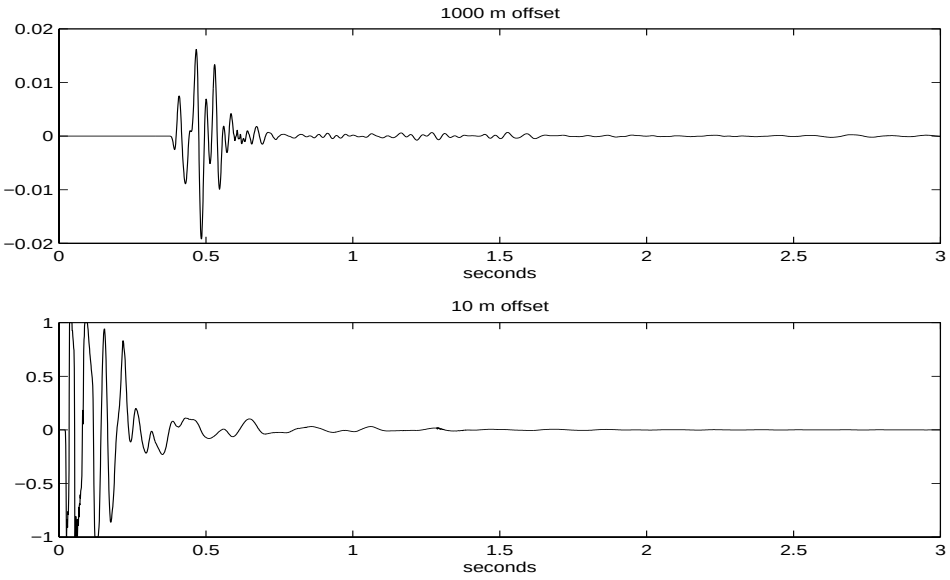

Figure 1.2: (Top) A real seismic trace recorded about 1000 m from a dynamite shot. (Bottom) A similar trace recorded only 10 m from the same shot. See Code Snippet 1.3.1

The close proximity of `tracenear` to a large explosion produces a very strong first arrival while later information (at 3 seconds) has decayed by ∼72 decibels. (To gain familiarity with decibel scales, it is useful to note that 6 db corresponds to a factor of 2. Thus 72 db represents about 72/6 ~ 12 doublings or a factor of $2^{12} = 4096$.) Alternatively, `tracefar` shows peak amplitudes that are 40db (a factor of $2^{6.7} \sim 100$) weaker than `tracenear`.

Code Snippet 1.3.1. *This code loads near and far offset test traces, computes the* Hilbert envelopes *of the traces (with a decibel scale), and produces Figures 1.2 and 1.3.*

```
1    clear; load testtrace.mat
2    subplot(2,1,1);plot(t,tracefar);
3    title('1000 m offset');xlabel('seconds')
4    subplot(2,1,2);plot(t,tracenear);
5    title('10 m offset');xlabel('seconds')
6    envfar = abs(hilbert(tracefar)); %compute Hilbert envelope
7    envnear = abs(hilbert(tracenear)); %compute Hilbert envelope
8    envdbfar=todb(envfar,max(envnear)); %decibel conversion
9    envdbnear=todb(envnear); %decibel conversion
10   figure
11   plot(t,[envdbfar envdbnear],'b');xlabel('seconds');ylabel('decibels');
12   grid;axis([0 3 -140 0])
```

End Code

The *first break time* is the best estimate of the arrival time of the first seismic energy. For `tracefar` this is about .380 seconds while for `tracenear` it is about .02 seconds. On each trace, energy before this time cannot have originated from the source detonation and is usually taken as

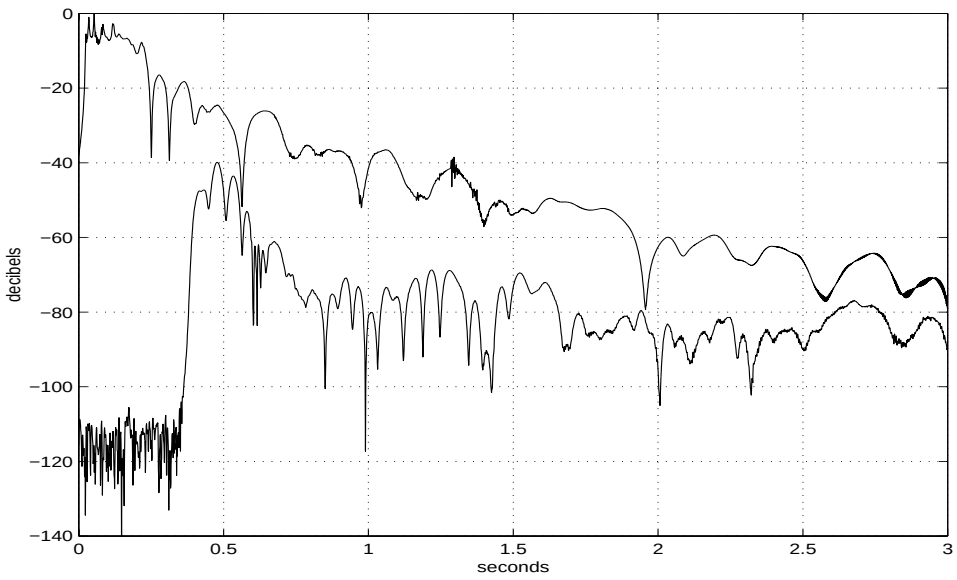

Figure 1.3: The envelopes of the two traces of Figure 1.2 plotted on a decibel scale. The far offset trace is about 40 db weaker than the near offset and the total dynamic range is about 120 db. See Code Snippet 1.3.1

an indication of ambient noise conditions. That is, it is due to seismic noise caused by wind, traffic, and other effects outside the seismic experiment. Only for `tracefar` is the first arrival late enough to allow a reasonable sampling of the ambient noise conditions. In this case, the average background noise level is about 120 to 130 db below the peak signals on `tracenear`. This is very near the expected instrument performance. It is interesting to note that the largest peaks on `tracenear` appear to have square tops, indicating clipping, at an amplitude level of 1.0. This occurs because the recording system gain settings were set to just clip the strongest arrivals and therefore distribute the available dynamic range over an amplitude band just beneath these strong arrivals.

Dynamic range is an issue in seismic display as well as recording. It is apparent from Figure 1.2 that either signal fades below visual thresholds at about 1.6 seconds. Checking with the envelopes on Figure 1.3, this suggests that the dynamic range of this display is about 40-50 db. This limitation is controlled by two factors: the total width allotted for the trace plot and the minimum perceivable trace deflection. In Figure 1.2 this width is about 1 inch and the minimum discernible wiggle is about .01 inches. Thus the dynamic range is about $10-2 \sim 40$db, in agreement with the earlier visual assessment.

It is very important to realize that seismic displays have limited dynamic range. This means that what-you-see is not always what-you've-got. For example if a particular seismic display being interpreted for exploration has a dynamic range of say 20 db, then any spectral components (i.e. frequencies in the Fourier spectrum) that are more than 20 db down will not affect the display. If these weak spectral components are signal rather than noise, then the display does not allow the optimal use of the data. This is an especially important concern for wiggle trace displays of multichannel data where each trace gets about 1/10 of an inch of display space.

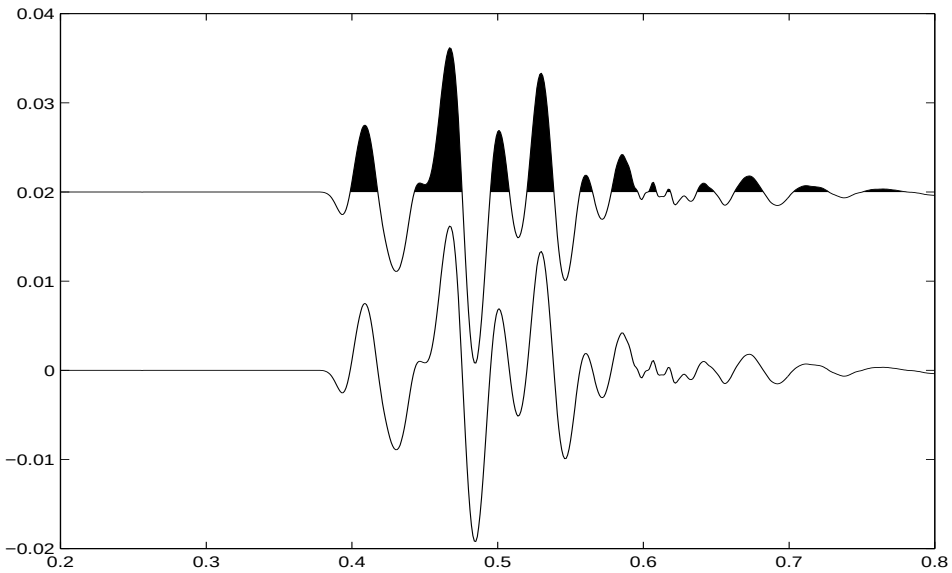

Figure 1.4: A portion of the seismic trace in Figure 1.2 is plotted in WTVA format (top) and WT format (bottom). See Code Snippet 1.3.2

More popular than the wiggle-trace display is the wiggle-trace, variable-area (WTVA) display. Function *wtva* (Code Snippet 1.3.2) was be used to create Figure 1.4 where the two display types are contrasted using `tracefar`. The WTVA display fills-in the *peaks* of the seismic trace (or *troughs* if polarity is reversed) with solid color. Doing this requires determining the *zero crossings* of the trace which can be expensive if precision is necessary. Function *wtva* just picks the sample closest to each zero crossing. For more precision, the final argument of *wtva* is a resampling factor which causes the trace to be resampled and then plotted. Also, *wtva* works like MATLAB 's low-level `line` in that it does not clear the figure before plotting. This allows *wtva* to be called repeatedly in a loop to plot a seismic section. The return values of *wtva* are MATLAB graphics handles for the "wiggle" and the "variable area" that can be used to further manipulate their graphic properties. (For more information consult your MATLAB reference.) The horizontal line at zero amplitude on the WTVA plot is not usually present in such displays and seems to be a MATLAB graphic artifact.

Code Snippet 1.3.2. *The same trace is plotted with* `wtva` *and* `plot`*. Figure 1.4 is the result.*

```
1    clear;load testtrace.mat
2    plot(t,tracefar)
3    [h,hva]=wtva(tracefar+.02,t,'k',.02,1,-1,1);
4    axis([.2 .8 -.02 .04])
```

End Code

Clipping was mentioned previously in conjunction with recording but also plays a role in display. Clipping refers to the process of setting all values on a trace that are greater than a clip level equal to that level. This can have the effect of moving the dynamic range of a plot display into the lower amplitude levels. Figure 1.5 results from Code Snippet 1.3.3 and shows the effect of plotting the

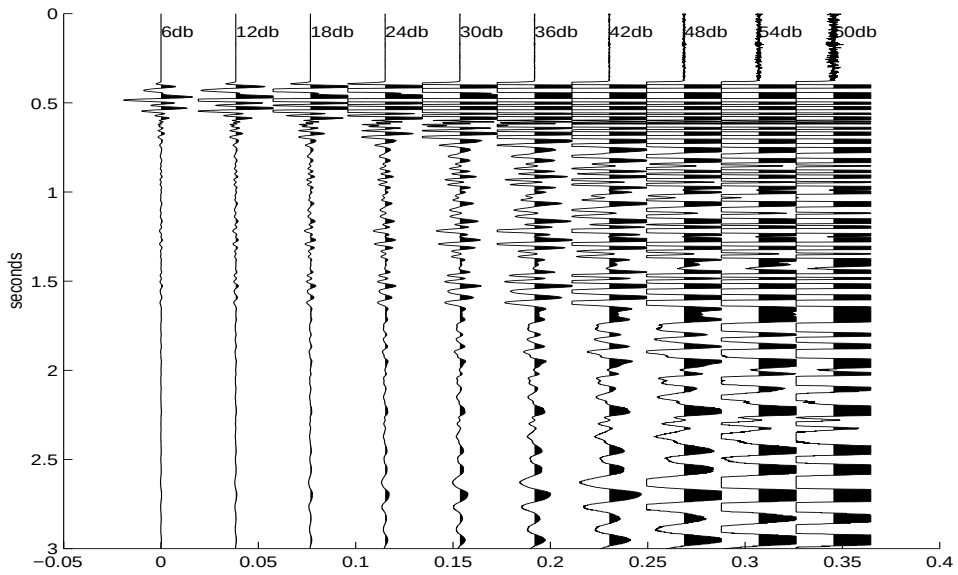

Figure 1.5: The seismic trace in Figure 1.2 is plotted repeatedly with different clip levels. The clip levels are annotated on each trace. See Code Snippet 1.3.3

same trace (`tracefar` at progressively higher clip levels. Function ***clip*** produces the clipped trace that is subsequently rescaled so that the clip level has the same numerical value as the maximum absolute value of the original trace. The effect of clipping is to make the weaker amplitudes more evident at the price of completely distorting the stronger amplitudes. Clipping does not increase the dynamic range, it just shifts the available range to a different amplitude band.

Code Snippet 1.3.3. *This code makes Figure 1.5. The test trace is plotted repeatedly with progressively higher clip levels.*

```
1    clear;load testtrace.mat
2    amax=max(abs(tracefar));
3    for k=1:10
4       trace_clipped=clip(tracefar,amax*(.5)^(k-1))/((.5)^(k-1));
5       wtva(trace_clipped+(k-1)*2*amax,t,'k',(k-1)*2*amax,1,1,1);
6       text((k-1)*2*amax,.1,[int2str(k*6) 'db']);
7    end
8    flipy;ylabel('seconds');
```

End Code

Exercise 1.3.1. *Use MATLAB to load the test traces shown in Figure 1.2 and display them. By appropriately zooming your plot, estimate the first break times as accurately as you can. What is the approximate velocity of the material between the shot and the geophone? (You may find the function* `simplezoom` *useful. After creating your graph, type* `simplezoom` *at the MATLAB prompt and then use the* left *mouse mutton to draw a zoom box. A double-click will un-zoom.)*

Exercise 1.3.2. *Use MATLAB to load the test traces shown in Figure 1.2 and compute the envelopes as shown in Code Snippet 1.3.1. For either trace, plot both the wiggle trace and its envelope and show that the trace is contained within the bounds defined by* $\pm$*envelope.*

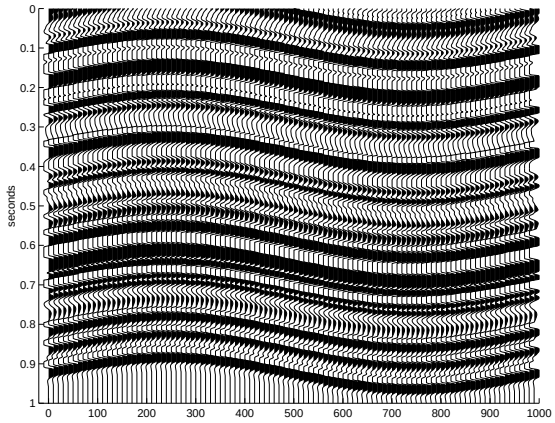

Figure 1.6: A synthetic seismic section plotted in WTVA mode with clipping. See Code Snippet 1.3.4

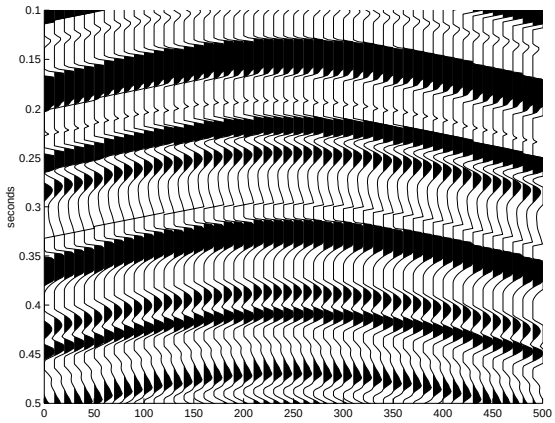

Figure 1.7: A zoom (enlargement of a portion) of Figure 1.6. Note the clipping indicated by square-bottomed troughs

Exercise 1.3.3. *What is the dynamic range of a seismic wiggle trace display plotted at 10 traces/inch? What about 30 traces/inch?*

1.3.2 Multichannel seismic display

Figure 1.5 illustrates the basic idea of a multichannel WTVA display. Assuming *ntr* traces, the plot width is divided into *ntr* equal segments to display each trace. A trace is plotted in a given plot segment by adding an appropriate constant to its amplitude. In the general case, these segments may overlap, allowing traces to over-plot one another. After a clip level is chosen, the traces are plotted such that an amplitude equal to the clip level gets a *trace excursion* to the edges of the trace plot segment. For hardcopy displays, the traces are usually plotted at a specified number per inch.

Figure 1.6 is a synthetic seismic section made by creating a single synthetic seismic trace and then replicating it along a sinusoidal (with x) trajectory. Figure 1.7 is a zoomed portion of the same synthetic seismic section. Function ***plotseis*** made the plot (see Code Snippet 1.3.3), and clipping was intentionally introduced. The square troughs on several events (e.g. near .4 seconds) are the signature of clipping. Function ***plotseis*** provides facilities to control clipping, produce either WT or WTVA displays, change polarity, and more.

Code Snippet 1.3.4. *Here we create a simple synthetic seismic section and plot it as a WTVA plot with clipping. Figure 1.6 is the result .*

```
1    global NOSIG;NOSIG=1;
2    [r,t]=reflec(1,.002,.2);%make reflectivity
3    nt=length(t);
4    [w,tw]=wavemin(.002,20,.2);%make wavelet
5    s=convm(r,w);%make convolutional seismogram
6    ntr=100;%number of traces
7    seis=zeros(length(s),ntr);%preallocate seismic matrix
8    shift=round(20*(sin([1:ntr]*2*pi/ntr)+1))+1; %a time shift for each trace
9    %load the seismic matrix
10   for k=1:ntr
11      seis(1:nt-shift(k)+1,k)=s(shift(k):nt);
```

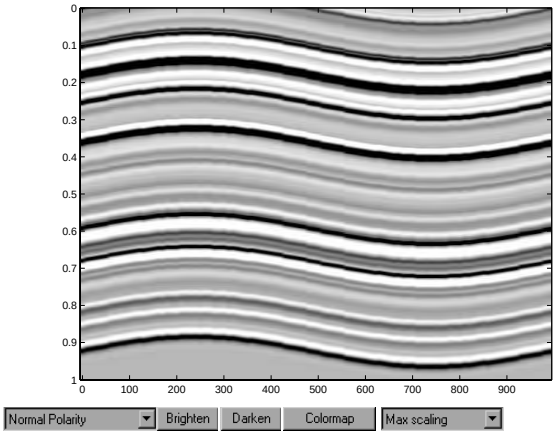

Figure 1.8: A synthetic seismic section plotted as an image using *plotimage*. Compare with Figure 1.6

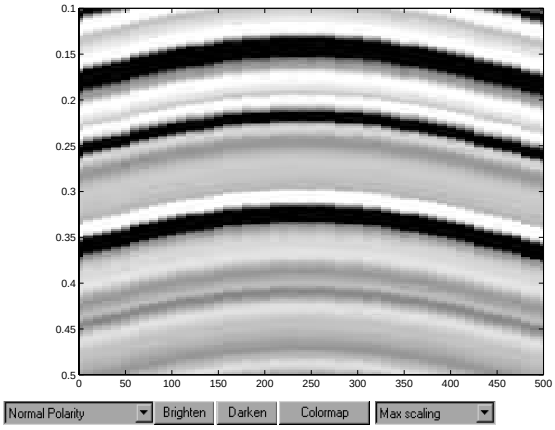

Figure 1.9: A zoom (enlargement of a portion) of Figure 1.8. Compare with Figure 1.7

```
12   end
13   x=(0:99)*10; %make an x coordinate vector
14   plotseis(seis,t,x,1,5,1,1,'k');ylabel('seconds')
```

End Code

Code Snippet 1.3.4 illustrates the use of global variables to control plotting behavior. The global `NOSIG` controls the appearance of a *signature* at the bottom of a figure created by *plotseis*. If `NOSIG` is set to zero, then *plotseis* will annotate the date, time, and user's name in the bottom right corner of the plot. This is useful in classroom settings when many people are sending nearly identical displays to a shared printer. The user's name is defined as the value of another global variable, `NAME_` (the capitalization and the underscore are important). Display control through global variables is also used in other utilities in the *NumMethToolbox* (see page 14). An easy way to ensure that these variables are always set as you prefer is to include their definitions in your *startup.m* file (see your MATLAB reference for further information).

The popularity of the WTVA display has resulted partly because it allows an assessment of the seismic waveform (if unclipped) as it varies along an event of interest. However, because its dynamic range varies inversely with trace spacing it is less suitable for small displays of densely spaced data such as on a computer screen. It also falls short for display of multichannel Fourier spectra where the wiggle shape is not usually desired. For these purposes, *image* displays are more suitable. In this technique, the display area is divided equally into small rectangles (pixels) and these rectangles are assigned a color (or gray-level) according to the amplitude of the samples within them. On the computer screen, if the number of samples is greater than the number of available pixels, then each pixel represents the average of many samples. Conversely, if the number of pixels exceeds the number of samples, then a single sample can determine the color of many pixels, leading to a blocky appearance.

Figures 1.8 and 1.9 display the same data as Figures 1.6 and 1.7 but used the function *plotimage* in place of *plotseis* in Code Snippet 1.3.4. The syntax to invoke *plotimage* is `plotimage(seis,t,x)`. It may also be called like `plotimage(seis)` in which case it creates x and t coordinate vectors as simply column and row numbers. Unlike *plotseis*, *plotimage* adorns its figure window with controls to allow the user to interactively change the polarity, brightness, color map, and data scaling

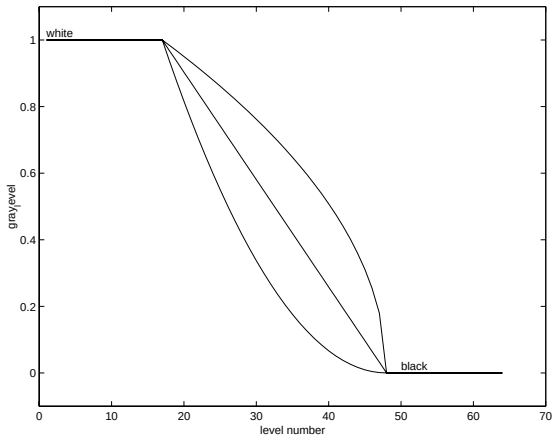

Figure 1.10: A 50% gray curve from *seisclrs* (center) and its brightened(top) and darkened (bottom) versions.

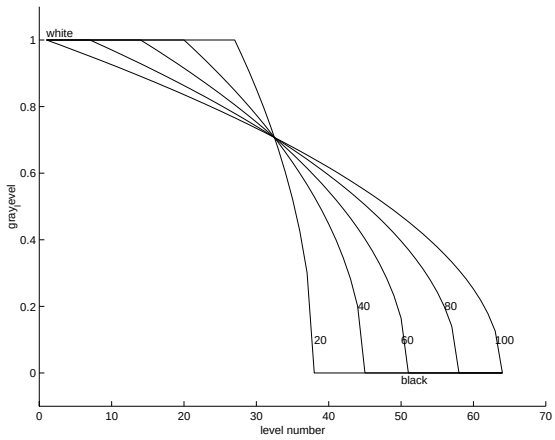

Figure 1.11: Various gray level curves from *seisclrs* for different `gray_pct` values.

scheme (though these controls are shown in these figures, in most cases in this book they will be suppressed). The latter item refers to the system by which amplitudes are mapped to gray levels (or colors).

Code Snippet 1.3.5. *This example illustrates the behavior of the* `seisclrs` *color map. Figures 1.10 and 1.11 are created.*

```
1    figure;
2    global NOBRIGHTEN
3
4    NOBRIGHTEN=1;
5    s=seisclrs(64,50); %make a 50% linear gray ramp
6    sb=brighten(s,.5); %brighten it
7    sd=brighten(s,-.5); %darken it
8    plot(1:64,[s(:,1) sb(:,1) sd(:,1)],'k')
9    axis([0 70 -.1 1.1])
10   text(1,1.02,'white');text(50,.02,'black')
11   xlabel('level number');ylabel('gray_level');
12   figure; NOBRIGHTEN=0;
13   for k=1:5
14      pct=max([100-(k-1)*20,1]);
15      s=seisclrs(64,pct);
16      line(1:64,s(:,1),'color','k');
17      if(rem(k,2))tt=.1;else;tt=.2;end
18      xt=near(s(:,1),tt);
19      text(xt(1),tt,int2str(pct))
20   end
21   axis([0 70 -.1 1.1])
22   text(1,1.02,'white');text(50,-.02,'black')
23   xlabel('level number');ylabel('gray_level');
```

End Code

By default, *plotimage* uses a gray level scheme, defined by *seisclrs*, which is quite nonlinear. Normally, 64 separate gray levels are defined and a linear scheme would ramp linearly from 1 (white)

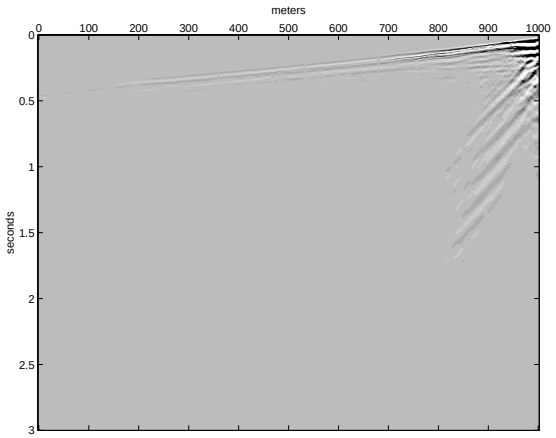

Figure 1.12: A raw shot record displayed using ***plotimage*** and *maximum scaling*

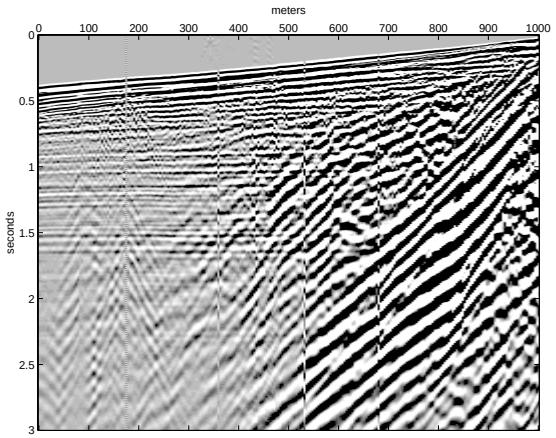

Figure 1.13: The same raw shot record as Figure 1.12 but displayed using *mean scaling* with a clip level (parameter CLIP) of 4.

at level 1 to 0 (black) at level 64. However, this was found to produce overly dark images with little event standout. Instead, ***seisclrs*** assigns a certain percentage, called `gray_pct`, of the 64 levels to the transition from white to black and splits the remaining levels between solid black and solid white. For example, by default this percentage is 50 which means that levels 1 through 16 are solid white, 17 through 48 transition from white to black, and 49 through 64 are solid black. The central curve in Figure 1.10 illustrates this. As a final step, ***seisclrs*** brightens the curve using `brighten` to produce the upper curve in Figure 1.10. Figure 1.11 shows the gray scales that result from varying the value of `gray_pct`.

Given a gray level or color scheme, function ***plotimage*** has two different schemes for determining how seismic amplitudes are mapped to the color bins. The simplest method, called *maximum scaling* determines the maximum absolute value in the seismic matrix, assigns this the color black, and the negative of this number is white. All other values map linearly into bins in between. This works well for synthetics, or well-balanced real data, but often disappoints for noisy or raw data because of the data's very large dynamic range. The alternative, called *mean scaling*, measures both the mean, $\bar{s}$ and standard deviation σ_s of the data and assigns the mean to the center of the gray scale. The ends of the scale are assigned to the values $\bar{s} \pm \text{CLIP}\sigma_s$ where CLIP is a user-chosen constant. Data values outside this range are assigned to the first or last gray-level bin thereby clipping them. Thus, mean scaling centers the gray scale on the data mean and the extremes of the scale are assigned to a fixed number of standard deviations from the mean. In both of these schemes, neutral gray corresponds to zero (if the ***seisclrs*** color map is used).

Figure 1.12 displays a raw shot record using the maximum scaling method with ***plotimage***. (The shot record is contained in the file *smallshot.mat*, and the traces used in Figure 1.2 are the first and last trace of this record.) The strong energy near the shot sets the scaling levels and is so much stronger than anything else that most other samples fall into neutral gray in the middle of the scale. On the other hand, figure 1.13 uses mean scaling (and very strong clipping) to show much more of the data character.

In addition to the user interface controls on its window, the behavior of plotimage can be controlled through *global variables*. Most variables defined in MATLAB are *local* and have a scope that

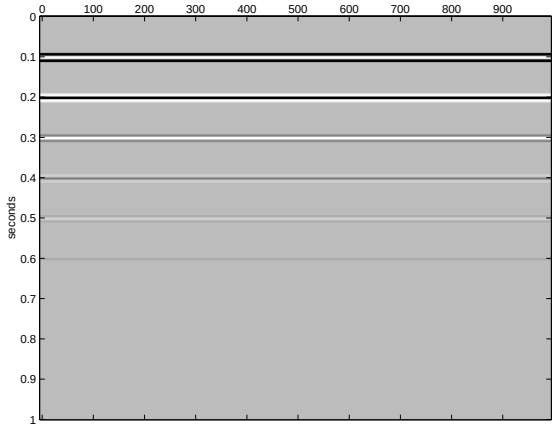

Figure 1.14: A synthetic created to have a 6db decrease with each event displayed by *plotimage*. See Code Snippet 1.3.7.

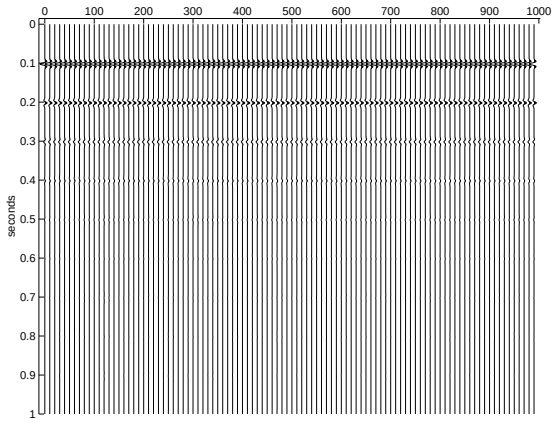

Figure 1.15: The same synthetic as Figure 1.14 displayed with *plotseis*.

is limited to the setting in which they are defined. For example, a variable called `x` defined in the base workspace (i.e. at the MATLAB prompt ≫) can only be referenced by a command issued in the base workspace. Thus, a function like *plotimage* can have its own variable `x` that can be changed arbitrarily without affecting the `x` in the base workspace. In addition, the variables defined in a function are transient, meaning that they are erased after the function ends. Global variables are an exception to these rules. Once declared (either in the base workspace or a function) they remain in existence until explicitly cleared. If, in the base workspace, `x` is declared to be global, then the `x` in *plotimage* is still independent unless *plotimage* also declares `x` global. Then, both the base workspace and *plotimage* address the same memory locations for `x` and changes in one affect the other.

Code Snippet 1.3.6 illustrates the assignment of global variables for *plotimage* as done in the author's startup.m file. These variables only determine the state of a *plotimage* window at the time it is launched. The user can always manipulate the image as desired using the user interface controls. Any such manipulations will not affect the global variables so that the next *plotimage* window will launch in exactly the same way.

Code Snippet 1.3.6. *This is a portion of the author's* startup.m *file that illustrates the setting of global variables to control* `plotimage`.

```
1    global SCALE_OPT GRAY_PCT NUMBER_OF_COLORS
2    global CLIP COLOR_MAP NOBRIGHTEN NOSIG
3    SCALE_OPT=2;
4    GRAY_PCT=20;
5    NUMBER_OF_COLORS=64;
6    CLIP=4;
7    COLOR_MAP='seisclrs';
8    NOBRIGHTEN=1;
9    NOSIG=1;
```

End Code

Code Snippet 1.3.7. *This code creates a synthetic with a 6db decrease for each event and displays it without clipping with both* `plotimage` *and* `plotseis`*. See Figures 1.14 and 1.15.*

```
% put an event every 100 ms. Each event decreases in amplitude by 6 db.
r=zeros(501,1);dt=.002;ntr=100;
t=(0:length(r)-1)*dt;
amp=1;
for tk=.1:.1:1;
   k=round(tk/dt)+1;
   r(k)=(-1)^(round(tk/.1))*amp;
   amp=amp/2;
end
w=ricker(.002,60,.1);
s=convz(r,w);
seis=s*ones(1,ntr);
x=10*(0:ntr-1);
global SCALE_OPT GRAY_PCT
SCALE_OPT=2;GRAY_PCT=100;;
plotimage(seis,t,x);ylabel('seconds')
plotseis(seis,t,x,1,1,1,1,'k');ylabel('seconds')
```

End Code

The dynamic range of image and WTVA displays is generally different. When both are displayed without clipping using a moderate trace spacing (Figures 1.14 and 1.15), the dynamic range of ***plotimage*** (≈30db) tends to be slightly greater than that of ***plotseis*** (≈20db). However, the behavior of ***plotimage*** is nearly independent of the trace density while ***plotseis*** becomes useless at high densities.

Exercise 1.3.4. *Load the file* `smallshot.mat` *into your base workspace using the command* ***load*** *and display the shot record with* `plotimage`*. Study the online help for* `plotimage` *and define the six global variables that control its behavior. Practice changing their values and observe the results. In particular, examine some of the color maps: hsv, hot, cool, gray, bone, copper, pink, white, flag, jet, winter, spring, summer, and autumn. (Be sure to specify the name of the color map inside single quotes.)*

Exercise 1.3.5. *Recreate Figures 1.14 and 1.15 using Code Snippet 1.3.7 as a guide. Experiment with different number of traces (*`ntr`*) and different program settings to see how dynamic range is affected.*

1.3.3 The *plotimage* picking facility

The function ***plotimage***, in addition to displaying seismic data, provides a rudimentary seismic *picking* facility. This is similar in concept to using MATLAB's `ginput` (described in the next section) but is somewhat simpler to use. In concept, this facility simply allows the user to draw any number of straight-line segments (picks) on top the the data display and affords easy access to the coordinates of these picks. The picks can then be used in any subsequent calculation or analysis.

The picking mechanism is activated by the popup menu in the lower left corner of the ***plotimage*** window. Initially in the *zoom* setting, this menu has two picking settings: *Pick(N)* and *Pick(O)*. The 'N' and 'O' stand for *New* and *Old* indicating whether the picks about to be made are a new list or should be added to the existing list. The pick list is stored as the global variable `PICKS` that can be accessed from the base workspace, or within any function, by declaring it there. There is only one pick list regardless of how many ***plotimage*** windows are open. This means that if one ***plotimage*** window has been used for picking and a second is then activated with the *Pick(N)* option, the picks

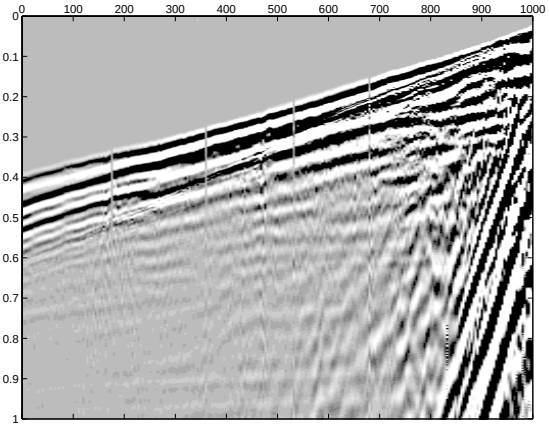

Figure 1.16: This figure is displayed by Code Snippet 1.3.8 before allowing the user to select points.

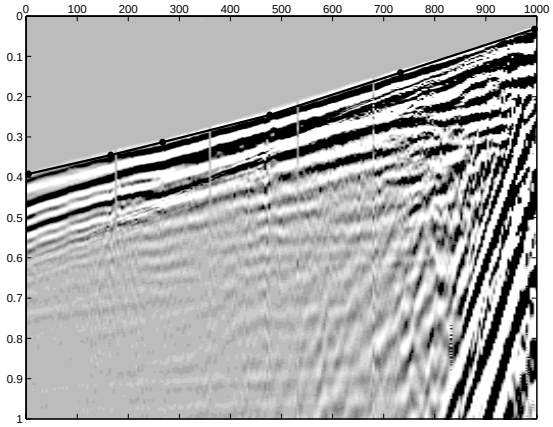

Figure 1.17: After the user picks the first breaks in Figure 1.16, the picks are drawn on top of the seismic data. See Code Snippet 1.3.8.

for the first window will be lost. If desired, this can be avoided by copying the pick list into another variable prior to activating the second picking process.

A pick is made by clicking the left mouse button down at the first point of the pick, holding the button down while dragging to the second point, and releasing the button. The pick will be drawn on the screen in a temporary color and then drawn in a permanent color when the mouse is released. The permanent color is controlled by the global variable `PICKCOLOR`. If this variable has not been set, picks are drawn in red. A single mouse click, made without dragging the mouse and within the bounds of the axes, will delete the most recent pick.

This picking facility is rudimentary in the sense that the picks are established without any numerical calculation involving the data. Furthermore, there is no facility to name a set of picks or to retain their values. Nevertheless, a number of very useful calculations, most notably raytrace migration (see sections 5.2.2 and 5.3.3) are enabled by this. It is also quite feasible to implement a more complete picking facility on this base.

1.3.4 Drawing on top of seismic data

Often it is useful to draw lines or even filled polygons on top of a seismic data display. This is quite easily done using MATLAB's `line` and `patch`commands. Only `line` is discussed here.

Code Snippet 1.3.8 loads the small sample shot record and displays it. The command `ginput` halts execution of the script and transfers focus to the current figure. There, the cursor will be changed to a crosshairs and the user is expected to enter a sequence of mouse clicks to define a series of points. In the example shown in Figures 1.16 and 1.17, six points were entered along the first breaks. `ginput` expects the points to be selected by simple mouse clicks and then the "enter" key is hit to signal completion. The six points are returned as a vector of x coordinates, `xpick`, and a vector of t coordinates, `tpick`. Though it is a useful tool, `ginput` does note provide a picking facility such as that described previously. However, `ginput` works with any MATLAB graphics display while the picking facility is only available with *`plotimage`*.

The call to `line` plots the points on top of the seismic image. The return value from `line` is the *graphics handle* of the line. The function `line` will accept vectors of the x, y, and possibly z

coordinates of the nodes of the line to be drawn. If no z coordinates are given, they are assumed zero which causes them to be drawn in the same z plane as the seismic (MATLAB graphics are always 3D even when they appear to be only 2D). Plotting the line at $z = 0$ usually works but if more mouse clicks are done on the figure, such as for zooming, then it can happen that the line may get re-drawn first with the seismic on top of it. This causes the line to "disappear". Instead, it is more robust to give the line a vector of z coordinates of unity so that it is guaranteed to always be in front of the seismic.

In addition to (x, y, z) coordinates, `line` accepts an arbitrary length list of (attribute, property) pairs that define various features of the line. In this case, its color is set to *red*, its line thickness is set to twice normal, and markers are added to each point. There are many other possible attributes that can be set in this way. To see a list of them, issue the command `get(h)` where `h` is the handle of a line and MATLAB will display a complete property list. Once the line has been drawn, you can alter any of its properties with the `set` command. For example, `set(h,'color','c')` changes the line color to cyan and `set(h,'ydata',get(h,'ydata')+.1)` shifts the line down .1 seconds.

Code Snippet 1.3.8. *This example displays the small sample shot record using* `plotimage` *and then uses* ***`ginput`*** *to allow the user to enter points. These points are plotted on top of the seismic data using* ***`line`****. The results are in Figures 1.16 and 1.17.*

```
1    load smallshot
2    global SCALE_OPT CLIP
3    SCALE_OPT=1;CLIP=1;
4    plotimage(seis,t,x)
5    axis([0 1000 0 1])
6    [xpick,tpick]=ginput;
7    h=line(xpick,tpick,ones(size(xpick)),'color','r',...
8       'linewidth',2,'marker','*');
```

End Code

1.4 Programming tools

It is recommended that a study course based on this book involve a considerable amount of programming. Programming in MATLAB has similarities to other languages but it also has its unique quirks. This section discusses some strategies for using MATLAB effectively to explore and manipulate seismic datasets. MATLAB has two basic programming constructs, *scripts* and *functions*, and *program* will be used to refer to both.

1.4.1 Scripts

MATLAB is designed to provide a highly interactive environment that allows very rapid testing of ideas. However, unless some care is taken, it is very easy to create results that are nearly irreproducible. This is especially true if a long sequence of complex commands has been entered directly at the MATLAB prompt. A much better approach is to type the commands into a text file and execute them as a *script*. This has the virtue that a permanent record is maintained so that the results can be reproduced at any time by simply re-executing the script.

A script is the simplest possible MATLAB programming structure. It is nothing more than a sequence of syntactically correct commands that have been typed into a file. The file name must end in .m and it must appear in the MATLAB search path. When you type the file name (without the .m) at the MATLAB prompt, MATLAB searches its path for the first .m file with that name and executes the commands contained therein. If you are in doubt about whether your script is the

first so-named file in the path, use the `which` command. Typing `which foo` causes MATLAB to display the complete path to the file that it will execute when `foo` is typed.

A good practice is to maintain a folder in MATLAB's search path that contains the scripts associated with each logically distinct project. These script directories can be managed by creating a further set of control scripts that appropriately manipulate MATLAB's search path. For example, suppose that most of your MATLAB tools are contained in the directory
`C:\Matlab\toolbox\local\mytools`
and that you have project scripts stored in
`C:\My Documents\project1\scripts`
and
`C:\My Documents\project2\scripts`.
Then, a simple management scheme is to include something like

```
1    global MYPATH
2    if(isempty(MYPATH))
3       p=path;
4       path([p ';C:\MatlabR11\toolbox\local\mytools']);
5       MYPATH=path;
6    else
7       path(MYPATH);
8    end
```

in your startup.m file and then create a script called project1.m that contains

```
1    p=path;
2    path([p ';C:\My Documents\project1\scripts'])
```

and a similar script for project2. When you launch MATLAB your startup.m establishes your base search path and assigns it to a global variable called MYPATH. Whenever you want to work on project1, you simply type `project1` and your base path is altered to include the project1 scripts. Typing `startup` again restores the base path and typing `project2` sets the path for that project. Of course you could just add all of your projects to your base path but then you cannot have scripts with the same name in each project.

Exercise 1.4.1. *Write a script that plots wiggle traces on top of an image plot. Test your script with the synthetic section of Code Snippet 1.3.4. (Hint:* `plotseis` *allows its plot to be directed to an existing set of axes instead of creating a new one. The "current" axes can be referred to with* `gca`*.)*

1.4.2 Functions

The MATLAB *function* provides a more advanced programming construct than the script. The latter is simply a list of MATLAB commands that execute in the base workspace. Functions execute in their own independent workspace that communicates with the base workspace through *input and output variables*.

The structure of a function

A simple function that clips a seismic trace is given in Code Snippet 1.4.1. This shows more detail than will be shown in other examples in this book in order to illustrate function documentation. The first line of `clip` gives the basic syntax for a function declaration. Since both scripts and functions are in .m files, it is this line that alerts MATLAB to the fact that clip is not a script. The function is declared with the basic syntax

```
[output_variable_list]=function_name(input_variable_list).
```

Rectangular brackets [. . .] enclose the list of output variables, while regular parentheses enclose the input variables. Either list can be of any length, and entries are separated by commas. For *clip*, there is only one output variable and the [. . .] are not required. When the function is invoked, either at the command line or in a program, a choice may be made to supply only the first n input variables or to accept only the first m output variables. For example, given a function definition like `[x1,x2]=foo(y1,y2)`, it may be legally invoked with any of

```
[a,b]=foo(c,d);
[a,b]=foo(c);
a=foo(c,d);
a=foo(c);
```

The variable names in the function declaration have no relation to those actually used when the function is invoked. For example, when `a=foo(c)` is issued the variable `c` becomes the variable (y1) within `foo`'s workspace and similarly, `foo`'s `x1` is returned to become the variable `a` in the calling workspace. Though it is possible to call `foo` without specifying `y2` there is no simple way to call it and specify `y2` while omitting `y1`. Therefore, it is advisable to structure the list of input variables such that the most important ones appear first. If an input variable is not supplied, then it must be assigned a *default* value by the function.

Code Snippet 1.4.1. *A simple MATLAB function to clip a signal.*

```
1    function trout=clip(trin,amp);
2    % CLIP performs amplitude clipping on a seismic trace
3    %
4    % trout=clip(trin,amp)
5    %
6    % CLIP adjusts only those samples on trin which are greater
7    % in absolute value than 'amp'. These are set equal to amp but
8    % with the sign of the original sample.
9    %
10   % trin= input trace
11   % amp= clipping amplitude
12   % trout= output trace
13   %
14   % by G.F. Margrave, May 1991
15   %
16   if(nargin~=2)
17      error('incorrect number of input variables');
18   end
19   % find the samples to be clipped
20    indices=find(abs(trin)>amp);
21   % clip them
22    trout=trin;
23    trout(indices)=sign(trin(indices))*amp;
```

End Code

In the example of the function *clip* there are two input variables, both mandatory, and one output variable. The next few lines after the function definition are all comments (i.e. non-executable) as is indicated by the % sign that precedes each line. The first contiguous block of comments following the function definition constitutes the *online help*. Typing `help clip` at the MATLAB prompt will cause these comment lines to be echoed to your MATLAB window.

The documentation for *clip* follows a simple but effective style. The first line gives the function name and a simple synopsis. Typing `help directory_name`, where directory_name is the name of a

directory containing many .m files, causes the first line of each file to be echoed giving a summary of the directory. Next appears one or more *function prototypes* that give examples of correct function calls. After viewing a function's help file, a prototype can be copied to the command line and edited for the task at hand. The next block of comments gives a longer description of the tasks performed by the function. Then comes a description of each input and output parameter and their defaults, if any. Finally, it is good form to put your name and date at the bottom. If your code is to be used by anyone other than you, this is valuable because it establishes who owns the code and who is responsible for fixes and upgrades.

Following the online help is the body of the code. Lines 16-18 illustrate the use of the automatically defined variable `nargin` that is equal to the number of input variables supplied at calling. *`clip`* requires both input variables so it aborts if `nargin` is not two by calling the function `error`. The assignment of *default values* for input variables can also be done using `nargin`. For example, suppose the `amp` parameter were to be allowed to default to 1.0. This can be accomplished quite simply with

```
if(nargin<2) amp=1; end
```

Lines 20, 22, and 23 actually accomplish the clipping. They illustrate the use of vector addressing and will be discussed more thoroughly in Section 1.5.1.

1.4.3 Coping with errors and the MATLAB debugger

No matter how carefully you work, eventually you will produce code with in it. The simplest errors are usually due to mangled syntax and are relatively easy to fix. A MATLAB language guide is essential for the beginner to decipher the syntax errors.

More subtle errors only become apparent at runtime after correcting all of the syntax errors. Very often, runtime errors are related to incorrect matrix sizes. For example, suppose we wish to pad a seismic matrix with `nzs` zero samples on the end of each trace and `nzt` zero traces on the end of the matrix. The correct way to do this is show in Code Snippet 1.4.2.

Code Snippet 1.4.2. *Assume that a matrix* `seis` *already exists and that* `nzs` *and* `nzt` *are already defined as the sizes of zero pads in samples and traces respectively. Then a padded seismic matrix is computed as:*

```
1    %pad with zero samples
2    [nsamp,ntr]=size(seis);
3    seis=[seis;zeros(nzs,ntr)];
4    %pad with zero traces
5    seis=[seis zeros(nsamp+nzs,nzt)];
```

End Code

The use of [. . .] on lines 3 and 5 is key here. Unless they are being used to group the return variables from a function, [. . .] generally indicate that a new matrix is being formed from the concatenation of existing matrices. On line 3, `seis` is concatenated with a matrix of zeros that has the same number of traces as `seis` but `nzs` samples. The semi-colon between the two matrices is crucial here as it is the *row separator* while a space or comma is the *column separator*. If the elements in [ ] on line 3 were separated by a space instead of a semi-colon, MATLAB would emit the error message:

```
??? All matrices on a row in the bracketed expression must have the same
number of rows.
```

This occurs because the use of the column separator tells MATLAB to put `seis` and `zeros(nxs,ntr)` side-by-side in a new matrix; but, this is only possible if the two items have the same number of rows. Similarly, if a row separator is used on line 5, MATLAB will complain about "`All matrices on a row in the bracketed expression must have the same number of columns`".

Another common error is an assignment statement in which the matrices on either side of the equals sign do not evaluate to the same size matrix. This is a fundamental requirement and calls attention to the basic MATLAB syntax: `MatrixA = MatrixB`. That is, the entities on both sides of the equals sign must evaluate to matrices of exactly the same size. The only exception to this is that the assignment of a constant to a matrix is allowed.

One rather insidious error deserves special mention. MATLAB allows variables to be "declared" by simply using them in context in a valid expression. Though convenient, this can cause problems. For example, if a variable called `plot` is defined, then it will mask the command `plot`. All further commands to plot data (e.g. `plot(x,y)`) will be interpreted as indexing operations into the matrix `plot`. More subtle, if the letter `i` is used as the index of a loop, then it can no longer serve its predefined task as $\sqrt{-1}$ in complex arithmetic. Therefore some caution is called for in choosing variable names, and especially, the common Fortran practice of choosing `i` as a loop index is to be discouraged.

The most subtle errors are those which never generate an overt error message but still cause incorrect results. These logical errors can be very difficult to eliminate. Once a program executes successfully, it must still be verified that it has produced the expected results. Usually this means running it on several test cases whose expected behavior is well known, or comparing it with other codes whose functionality overlaps with the new program.

The MATLAB debugger is a very helpful facility for resolving run-time errors, and it is simple to use. There are just a few essential debugger commands such as: `dbstop, dbstep, dbcont, dbquit, dbup, dbdown`. A debugging session is typically initiated by issuing the `dbstop` command to tell MATLAB to pause execution at a certain line number, called a *breakpoint*. For example `dbstop at 200 in plotimage` will cause execution to pause at the executable line nearest line 200 in *`plotimage`*. At this point, you may choose to issue another debug command, such as `dbstep` which steps execution a line at a time, or you may issue any other valid MATLAB command. This means that the entire power of MATLAB is available to help discover the problem. Especially when dealing with large datasets, it is often very helpful to issue plotting commands to graph the intermediate variables.

As an example of the debugging facility, consider the problem of extracting a *slice* from a matrix. That is, given a 2D matrix, and a trajectory through it, extract a sub-matrix consisting of those samples within a certain half-width of the trajectory. The trajectory is defined as a vector of row numbers, one-per-column, that cannot double back on itself. A first attempt at creating a function for this task might be like that in Code Snippet 1.4.3.

Code Snippet 1.4.3. *Here is some code for slicing through a matrix along a trajectory. Beware, this code generates an error.*

```
1    function s=slicem(a,traj,hwid)
2
3    [m,n]=size(a);
4    for k=1:n %loop over columns
5            i1=max(1,traj(k)-hwid); %start of slice
6            i2=min(m,traj(k)+hwid); %end of slice
7            ind=(i1:i2)-traj(k)+hwid; %output indices
8            s(ind,k) = a(i1:i2,k); %extract the slice
9    end
```

End Code

First, prepare some simple data like this:

```
» a=((5:-1:1)')*(ones(1,10))

a =

     5     5     5     5     5     5     5     5     5     5
     4     4     4     4     4     4     4     4     4     4
     3     3     3     3     3     3     3     3     3     3
     2     2     2     2     2     2     2     2     2     2
     1     1     1     1     1     1     1     1     1     1

» traj=[1:5 5:-1:1]
traj =
     1     2     3     4     5     5     4     3     2     1
```

The samples of `a` that lie on the trajectory are `5 4 3 2 1 1 2 3 4 5`. Executing `slicem` with the command `s=slicem(a,traj,1)` generates the following error:

```
??? Index into matrix is negative or zero.

Error in ==> slicem.m On line 8  ==>  s(ind,k) = a(i1:i2,k);
```

This error message suggests that there is a problem with indexing on line 8; however, this could be occurring either in indexing into `a` or `s`. To investigate, issue the command "`dbstop at 8 in slicem`" and re-run the program. Then MATLAB stops at line 8 and prints

```
 8       s(ind,k) = a(i1:i2,k);
 K»
```

Now, suspecting that the index vector `ind` is at fault we list it to see that it contains the values `1 2` and so does the vector `i1:i2`. These are legal indices. Noting that line 8 is in a loop, it seems possible that the error occurs on some further iteration of the loop (we are at k=1). So, execution is resumed with the command `dbcont`. At k=2, we discover that `ind` contains the values `0 1 2` while `i1:i2` is `1 2 3`. Thus it is apparent that the vector `ind` is generating an illegal index of `0`. A moments reflection reveals that line 7 should be coded as `ind=(i1:i2)-traj(k)+hwid+1;` which includes an extra `+1`. Therefore, we exit from the debugger with `dbquit`, make the change, and re-run *slicem* to get the correct result:

```
s =

     0     5     4     3     2     2     3     4     5     0
     5     4     3     2     1     1     2     3     4     5
     4     3     2     1     0     0     1     2     3     4
```

The samples on the trajectory appear on the central row while the other rows contain neighboring values unless such values exceed the bounds of the matrix `a`, in which case a zero is returned. A more polished version of this code is found as *slicemat* in the *NumMethToolbox*. Note that a simpler debugger procedure (rather than stopping at line 8) would be to issue the command "`dbstop if error`" but the process illustrated here shows more debugging features.

1.5 Programming for efficiency

1.5.1 Vector addressing

The actual trace clipping in Code Snippet 1.4.1 is accomplished by three deceptively simple lines. These lines employ an important MATLAB technique called *vector addressing* that allows arrays to be processed without explicit loop structures. This is a key technique to master in order to write efficient MATLAB code. Vector addressing means that MATLAB allows an index into an array to be a vector (or more generally a matrix) while languages like C and Fortran only allow scalar indices. So, for example, if `a` is a vector of length 10, then the statement `a([3 5 7])=[pi 2*pi sqrt(pi)]` sets the third, fifth, and seventh entries to $\pi, 2 * \pi,$ and$\sqrt{\pi}$ respectively. In ***clip*** the function `find` performs a logical test and returns a *vector of indices* pointing to samples that satisfy the test. The next line creates the output trace as a copy of the input. Finally, the last line uses vector addressing to clip those samples identified by `find`.

Code Snippet 1.5.1. *A Fortran programmer new to MATLAB would probably code* `clip` *in this way.*

```
1    function trout=fortclip(trin,amp)
2
3    for k=1:length(trin)
4       if(abs(trin(k)>amp))
5          trin(k)=sign(trin(k))*amp;
6       end
7    end
8
9    trout=trin;
```

End Code

Code Snippet 1.5.2. *This script compares the execution time of* `clip` *and* `fortclip`

```
1    [r,t]=reflec(1,.002,.2);
2
3    tic
4    for k=1:100
5       r2=clip(t,.05);
6    end
7    toc
8
9    tic
10   for k=1:100
11      r2=fortclip(t,.05);
12   end
13   toc
```

End Code

In contrast to the vector coding style just described Code Snippet 1.5.1 shows how clip might be coded by someone stuck-in-the-rut of scalar addressing (a C or Fortran programmer). This code is logically equivalent to ***clip*** but executes much slower. This is easily demonstrated using MATLAB's built in timing facility. Code Snippet 1.5.2 is a simple script that uses the `tic` and `toc` commands for this purpose. `tic` sets an internal timer and `toc` writes out the elapsed time since the previous `tic`. The loops are executed 100 times to allow minor fluctuations to average out. On the second execution of this script, MATLAB responds with

```
elapsed_time = 0.0500
elapsed_time = 1.6000
```

which shows that *clip* is 30 times faster than *fortclip*. (On the first execution, *clip* is only about 15 times faster because MATLAB spends some time doing internal compiling. Run time tests should always be done a number of times to allow for effects like this.)

A simple blunder that can slow down *fortclip* even more is to write it like

Code Snippet 1.5.3. *An even slower version of* `fortclip`

```
1    function trout=fortclip(trin,amp)
2
3    for k=1:length(trin)
4       if(abs(trin(k)>amp))
5          trout(k)=sign(trin(k))*amp;
6       end
7    end
```

End Code

This version of *fortclip*, still produces correct results, but is almost 50 times slower than *clip*. The reason is that the output trace, `trout`, is being addressed sample-by-sample but it has not been pre-allocated. This forces MATLAB to resize the vector each time through the loop. Such resizing is slow and may require MATLAB to make repeated requests to the operating system to grow its memory.

Exercise 1.5.1. *Create a version of* `fortclip` *and verify the run time comparisons quoted here. Your computer may give different results. Show that the version of* `fortclip` *in Code Snippet 1.5.3 can be made to run as fast as that in Code Snippet 1.5.1 by using the* `zeros` *function to pre-allocate* `trout`.

1.5.2 Vector programming

It is this author's experience that MATLAB code written using vector addressing and linear algebra constructs can be quite efficient. In fact run times can approach that of compiled C or Fortran. On the other hand, coding in the scalar style can produce very slow programs that give correct results but lead to the false impression that MATLAB is a slow environment. MATLAB is designed to eliminate many of the loop structures found in other languages. Vector addressing helps in this regard but even more important is the use of MATLAB's linear algebra constructs. Programming efficiently in this way is called *vector programming*.

As an example, suppose `seis` is a seismic data matrix and it is desired to scale each trace (i.e. column) by a constant scale factor. Let `scales` be a vector of such factors whose length is equal to the number of columns in `seis`. Using scalar programming, a seasoned Fortran programmer might produce something like Code Snippet 1.5.4.

Code Snippet 1.5.4. *A Fortran programmer might write this code to scale each trace in the matrix* `seis` *by a factor from the vector* `scales`

```
1    [nsamp,ntr]=size(seis);
2
3    for col=1:ntr
4       for row=1:nsamp
5          seis(row,col)=scales(col)*seis(row,col);
6       end
7    end
```

End Code

This works correctly but is needlessly slow. An experienced MATLAB programmer would know that the operation of 'matrix times diagonal matrix' results in a new matrix whose columns are scaled by the corresponding element of the diagonal matrix. You can check this out for yourself by a simple MATLAB exercise:

```
a=ones(4,3); b=diag([1 2 3]); a*b
```

to which MATLAB's response is

```
ans =

     1     2     3
     1     2     3
     1     2     3
     1     2     3
```

Thus the MATLAB programmer would write Code Snippet 1.5.4 with the very simple single line in Code Snippet 1.5.5.

Code Snippet 1.5.5. *A MATLAB programmer would write the example of Code Snippet 1.5.4 in this way:*

```
1    seis=seis*diag(scales);
```

End Code

Tests indicate that Code Snippet 1.5.5 is about ten times as fast as Code Snippet 1.5.4.

Vector programming is also facilitated by the fact that MATLAB's mathematical functions are automatically set up to operate on arrays. This includes functions like `sin, cos, tan, atan, atan2, exp, log, log10, sqrt` and others. For example, if `t` is a time coordinate vector then `sin(2*pi*10*t)` is a vector of the same size as `t` whose entries are a 10 Hz. sine wave evaluated at the times in `t`. These functions all perform element-by-element operations on matrices of any size. Code written in this way is actually more visually similar to the corresponding mathematical formulae than scalar code with its many explicit loops.

Some of MATLAB's vectorized functions operate on matrices but return matrices of one dimension smaller. The functions `sum, cumsum, mean, main, max, std` all behave in this manner. For example, if `trace` is a vector and `seis` is a matrix, then `mean(trace)` results in a scalar that is the mean value of the vector while `mean(seis)` results in a *row vector* containing the mean value of each *column* of `seis`. The mean value of the entire matrix, `seis`, can be calculated with `mean(mean(seis))`.

1.5.3 The COLON symbol in MATLAB

Vector programming is also facilitated by a thorough understanding of the multiple uses of the colon (:) in MATLAB. In its most elementary form the colon is used to generate vectors. For example, the time coordinate vector for a trace with `ntr` samples and a sample interval of `dt` can be generated with the command `t=(0:ntr-1)*dt`. This results in a *row vector* whose entries are `[0 dt 2*dt 3*dt ... (ntr-1)*dt]`.

More subtle is the use of the colon in indexing into a matrix. Let `seis` be a two dimensional seismic matrix, then some sample indexing operations are:

`seis(:,10)` refers to the 10^{th} trace. The colon indicates that all rows (samples) are desired.

`seis(:,50:60)` refers to traces 50 through 60.

`seis(100:500,50:60)` selects samples 100 through 500 on traces 50 through 60.

`seis(520:2:720,:)` grabs very other sample between sample number 520 and number 720 on all traces.

`seis(:,90:-1:50)` selects traces 50 through 90 but in reverse order.

`seis(:)` refers to the entire seismic matrix as a giant column vector.

Of course, the indices need not be entered as explicit numbers but could be a variable instead as in the case of the return from `find` in Code Snippet 1.4.1.

The last item needs more explanation. Regardless of the dimensions of a matrix, it can always be referred to as a single column vector using a single index. For example, the two dimensional matrix, `seis`, can be indexed like `seis(irow,icol)` and as `seis(ipos)`. This is possible because computer memory is actually a one-dimensional space and MATLAB stores a matrix in this space in *column order*. This means that the actual storage pattern of `seis` is `[seis(1,1) seis(2,1) ...seis(nrows,1) seis(1,2) seis(2,2) ... ...seis(nrows,ncols)]`. Thus the last sample of column one and the first sample of column two are contiguous in memory. This means that the first sample of column two can be addressed either by `seis(1,2)` or by `seis(nrows+1)`. In accordance with this formulation, MATLAB allows any matrix to be entered into a formula as a equivalent column vector using the notation `seis(:)`.

1.5.4 Special values: NaN, Inf, and eps

MATLAB supports the IEEE representations of NaN (Not-a-Number) and Inf (Infinity). Certain mathematical operations can cause these to arise and they then behave in defined ways. For example `0/0, Inf*0, Inf±Inf, and Inf/Inf` all result in NaN. Furthermore, any arithmetic operator involving a NaN is required to produce NaN as an output. This can cause unexpected results. For example, `max([1 2 3 NaN 4 5 6])` results in NaN and the functions `min, mean, std` and many others will also return NaN on this vector. Even worse, running `fft` on data with only a single NaN in it will produce a NaN for the entire output spectrum. Despite these apparent complications, NaN's have enough uses that they are well worth having around. The occurrence of a NaN in your output data signals you that an unexpected arithmetic exception has taken place in your code. This must be found and corrected. NaN's can also be used as convenient place-holders to mark some special point in a matrix because they can easily be found. Also, when a NaN occurs in graphics, the action is to blank that part of the display. Thus NaN's can be used to temporarily turn off parts of a graphic, something that is especially useful in 3D.

When used in logical comparisons, such as `NaN==5`, the result is almost always False. This means that an expression such as `find(v==NaN)` will not correctly identify any NaN's in the vector `v`. (Of course, it works just fine if you want to find a number such as `find(v==5)`.) So, NaN's must be located using `find(isnan(v))` that returns a vector of indices of any NaN's in `v`. (The function `isnan` returns a logical vector of 1's and 0's indicating vector elements are NaN's.)

Infinity is handled similarly to NaN's in that certain operations generate infinity that subsequently behaves in a defined way. For example `x/0` generates an Inf and `x/Inf` always returns zero (unless x is Inf when NaN results). Inf also must be handled specially in logical comparisons. The functions `isinf` and `isfinite` can be used by themselves or as input to `find` as discussed above

with `isnan`. Also, the debug command `dbstop if naninf` is helpful to stop execution at the point that NaN or Inf occurs in any function.

MATLAB uses double precision floating point arithmetic for its calculations. This is actually a higher standard of precision than has traditionally been used in seismic data processing. Most processing systems have been written in Fortran or C and use single precision arithmetic. MATLAB supplies the internal constant `eps` whose value is the smallest positive floating point number available on your machine. In a loose sense, `eps` represents the machine "noise level" and is about $2.2204\text{x}10^{-16}$ on the author's computer. (How many decibels down from unity is eps?) As a general rule, tests for the equality of two floating point variables should be avoided. Instead of a test like `if(x==y)` consider something like `if(abs(x-y)<10*eps)`. `eps` can also be useful in computing certain limits that involve a division by zero. For example, computing a sinc function with `x=0:.1:pi;y=sin(x)./x;` generates a zero divide and `y(1)` is NaN. However, `x=0:.1:pi;y=sin(x+eps)./(x+eps);` avoids the zero divide and results in `y(1)` of 1.

1.6 Chapter summary

The scope of this book was defined as a survey of the fundamental numerical methodologies used in seismic exploration, and their physical basis. Reasons for the choice of the MATLAB language for the presentation of algorithms were given. MATLAB was suggested as a superior environment due to its vector language, extensive numerical toolbox, interactive environment, and built-in graphics. MATLAB conventions for naming functions were explained and the use of numbered Code Snippets in a theorem-like environment code presentation was discussed.

Considerable time was spent discussing the display of seismic data. It was argued that the numerical range (dynamic range) of modern data is very large, and this leads to the likelihood of clipping in display. Wiggle trace, WTVA, and image plots were discussed as display tools, and their relative merits were contrasted.

Finally, a brief survey of MATLAB programming was presented, with an emphasis on topics of importance when dealing with large datasets. Scripts and functions were discussed and their structure outlined. Common programming blunders that lead to reduced performance were discussed. Vector addressing, vector programming, and the *colon* operator were discussed with an emphasis on how they relate to the creation of efficient code.

Chapter 2

Signal Theory

The theory of digital signal processing is a vast subject and its complete description would require much more space than the signal chapter available here. However, an acquaintance with the essential elements of the theory is so essential that this chapter is included as an introduction. The scope of the present discussion is limited to fundamentals that have direct relevance to later chapters. Those requiring a more thorough exposition should consult a textbook such as Karl (1989).

As a general theme, concepts will be developed here for both continuous and sampled signals. The continuous viewpoint arises naturally when developing models for physical effects where the solutions to the relevant partial differential equations may be regarded as continuous signals. Alternatively, when solving such equations in complex situations or in processing real seismic data, digital computations in a computer are the only practical approach. Therefore, both approaches will be explored where appropriate and the transformation of a continuous expression into a discrete one, and the reverse, will be demonstrated.

2.1 Convolution

The process of convolution is so fundamental to all of signal theory that its description is an excellent starting point. A variety of descriptions of convolution are given in this section. They are all useful in different contexts and some will appeal more to the intuition than others. Fundamentally however, these are all merely different perspectives on the same mathematical expression that is stated as equation (2.3) below.

2.1.1 Convolution as filtering

Convolution is the name given to the process of applying a *stationary* linear filter to a signal. By stationary, it is meant that the filter response to an input impulse is independent of the time of the impulse. Another name for this property of stationarity is *translation invariance*. The filter is linear because the response to an input $a(t) + b(t)$ is the sum of the responses to $a(t)$ and $b(t)$ taken separately. Such a linear, stationary filter is completely specified by its impulse response, $f(t)$. This is defined as the output of the filter when the input is a unit impulse at $t = 0$. Often the phrase *the filter* will be used as a synonym for *the impulse response of the filter*. Suppose the input to the filter is an impulse of magnitude a_0 at time t_0 and a second impulse of magnitude a_1 at time t_1. Then the filter output is

$$g(t) = a_0 f(t - t_0) + a_1 f(t - t_1). \tag{2.1}$$

The property of linearity has been used to form the scaled sum of two time-shifted copies of the impulse response. These appear in the equation as $f(t-t_k)$ where $k=0$ or $k=1$. Thus $f(t=0)$ appears at both $t=t_0$ and $t=t_1$. Filters that represent physical systems have the *causality* property that $f(t)=0$ for $t<0$. If $f(t)$ is causal, then equation (2.1) places the first value that is possibly nonzero at the times of the input impulses. The property of stationarity has been used in that the response to the input impulse at $t=t_1$ is the same function $f(t)$ as for the impulse at $t=t_0$.

Suppose the input to the filter is an arbitrary sequence of impulses defined by the magnitudes $\{a_k\}, k = 1, 2, \ldots N$ and the corresponding times $\{t_k\}$. Then equation (2.1) generalizes to the one-dimensional convolution integral

$$g(t) = \sum_{k=1}^{n} a_k f(t-t_k). \tag{2.2}$$

Both this equation and equation (2.1) are examples of convolution equations though even equation (2.2) assumes too much about the nature of the input to be considered general. The defining feature is the formation of a superposition of scaled-and-delayed copies of a single function, $f(t)$. If a completely general expression is desired, then it must accommodate an input that is a continuous function of time rather than a sequence of impulses. In a limiting procedure, as the sequence of input impulses approaches the continuous function $a(t)$, equation (2.2) approaches

$$g(t) = \int_{-\infty}^{\infty} a(\tau) f(t-\tau) d\tau. \tag{2.3}$$

Here, in comparison with equation (2.2), an integration has replaced the summation, the integration variable τ has taken the place of the summation index k, and the sequence of impulses $\{a_k\}$ has become the continuous function $a(\tau)$. The presence of the dt time differential in equation (2.3) means that it is superficially dimensionally inconsistent with equation (2.2). This may be remedied either by assuming that $\{a_k\}$ and $a(t)$ have different units or, more commonly, by multiplying the right-hand-side of equation (2.2) by a constant with temporal dimensions. The limits of integration are formally written from $-\infty$ to ∞ to accommodate all possible input signals. If $a(\tau)$ is causal, then the lower limit may be altered to zero. If $a(\tau)$ vanishes for $\tau > \tau_{max}$ then the upper limit may be set to τ_{max}. In other words, the integration extends over all relevant values of τ.

Convolution arises in many parts of physical theory other than the application of linear, stationary filters. In all cases, the interpretation given here of forming a superposition of scaled and time-shifted copies of a physical response (or impulse response) is appropriate. However, the choice of which function, a or f, to interpret as the impulse response is arbitrary. If the integral operation of equation (2.3) is symbolized abstractly as $g = a \bullet f$, then it is elementary to show that $a \bullet f = f \bullet a$. This is accomplished by the change of variables $\tau' = t - \tau$ that gives

$$g(t) = \int_{-\infty}^{\infty} a(\tau) f(t-\tau) d\tau = \int_{\infty}^{-\infty} a(t-\tau') f(\tau') d(-\tau') = \int_{-\infty}^{\infty} a(t-\tau') f(\tau') d\tau'. \tag{2.4}$$

Thus convolution is said to be commutative and either function can be interpreted as the impulse response with the other being the input signal that provides the weighting.

Now consider the digital sampling of all three of the functions involved in equation (2.3). Let Δt be the *temporal sample interval* and let each of $a(t), f(t),$ and $g(t)$ be represented by sequences of discrete samples $\{a_k\}$, $\{f_k\}$, and $\{g_k\}$ with $k = 0, 1, \ldots N-1$ so that $t = k\Delta t$ ranges from 0 to $(N-1)\Delta t$. The difference between these sequences and that defined previously for equation (2.2) is

that these new sequences are all sampled at a regular interval. In the previous instance, a sequence of five impulses required only five samples regardless of the intervals between the samples. Now, if any of the five impulses are separated by an interval greater than Δt then more than five samples will be needed though the additional samples will be zero. At this expense of possibly including many zero samples, a *discrete* version of equation (2.3) can be written as

$$g_j = \Delta t \sum_{k=0}^{N-1} a_k f_{j-k} \tag{2.5}$$

where the constant Δt has been taken outside the summation. In most implementations of digital signal theory, Δt is a constant throughout the application and it is customary to write equations like (2.5) with Δt set to unity. Thus, more commonly discrete convolution will be written

$$g_j = \sum_{k=0}^{N-1} a_k f_{j-k}. \tag{2.6}$$

Exercise 2.1.1. *Show that convolution is distributive. That is $a \bullet (b + c) = a \bullet b + a \bullet c$. Is it better to stack (sum) a series of traces and then filter the stacked trace or to filter each one and stack the filtered traces? Why?*

Exercise 2.1.2. *Let $u(t) = \exp(i\omega t)$. For an arbitrary function $v(t)$ show that the convolution $u \bullet v = a(\omega)u$ where $a(\omega)$ is a function of frequency and of v. Determine the precise form of a to show how it depends on v.*

Exercise 2.1.3. *Study the example below and then compute the convolution of $r = [1, -.3, 0, -.5, .8, -.8]$ with $w = [.1, .3, .1]$. Make a sketch of the process of reversing w and sliding it past r. Redraw this sketch for each output sample.*

Example 2.1.1. *A simple model of a seismic trace, s_k is that it represents a reflectivity series, r_k convolved with a wavelet, w_k. That is $s_j = \sum_k r_k w_{j-k}$. Let $r = [1, .2, -.1, -.5, .5, 0, -1]$ and $w = [1, -.5, .2,]$ and compute s_j. Assuming that both $\{r\}$ and $\{w\}$ begin with sample number 0, s_0 is given by $s_0 = \sum_k r_k w_{0-k}$. The only nonzero contribution to this sum is for $k = 0$ with the result $s_0 = r_0 w_0 = 1$. Below, each element of $\{s\}$ is worked out. In each case, the limits on the summation are adjusted to include only the terms that involve nonzero contributions from $\{w\}$. This means that a maximum of three elements occur in each sum. The entire process can be visualized by reversing*

$\{w\}$ and sliding it past $\{r\}$. At each position, the samples that align are multiplied and summed.

$$s_0 = \sum_{k=0}^{0} r_k w_{0-k} = r_0 w_0 = 1$$

$$s_1 = \sum_{k=0}^{1} r_k w_{1-k} = r_0 w_1 + r_1 w_0 = -.5 + .2 = -.3$$

$$s_2 = \sum_{k=0}^{2} r_k w_{2-k} = r_0 w_2 + r_1 w_1 + r_2 w_0 = .2 - .1 - .1 = 0$$

$$s_3 = \sum_{k=1}^{3} r_k w_{3-k} = r_1 w_2 + r_2 w_1 + r_3 w_0 = .04 + .05 - .5 = -.41$$

$$s_4 = \sum_{k=2}^{4} r_k w_{4-k} = r_2 w_2 + r_3 w_1 + r_4 w_0 = -.02 + .25 + .5 = .73$$

$$s_5 = \sum_{k=3}^{5} r_k w_{5-k} = r_3 w_2 + r_4 w_1 + r_5 w_0 = -.1 - .25 + 0 = -.35$$

$$s_6 = \sum_{k=4}^{6} r_k w_{6-k} = r_4 w_2 + r_5 w_1 + r_6 w_0 = .1 + 0 - 1 = -.9$$

$$s_7 = \sum_{k=5}^{6} r_k w_{7-k} = r_5 w_2 + r_6 w_1 = 0 + .5 = .5$$

$$s_8 = \sum_{k=6}^{6} r_k w_{8-k} = r_6 w_2 = -.2$$

2.1.2 Convolution by polynomial multiplication - the Z transform

There is a convenient representation of discretely sampled signals that allows convolution to be done through a simple process of multiplication. Suppose that an N-length sequence $\{a\} = [a_0, a_1, a_2, \ldots a_{N-1}]$ is identified with a polynomial in a variable z by the relation

$$a(z) = a_0 + a_1 z + a_2 z^2 + \ldots a_{N-1} z^{N-1} = \sum_{k=0}^{N-1} a_k z^k. \tag{2.7}$$

The resulting function, $a(z)$ is called the *z transform* of the sequence $\{a\}$. Now, recall the rule for the multiplication of two polynomials, for example, let N=3 and multiply

$$\begin{aligned} a(z)f(z) &= \left[a_0 + a_1 z + a_2 z^2\right]\left[f_0 + f_1 z + f_2 z^2\right] \\ &= a_0 f_0 + a_0 f_1 z + a_0 f_2 z^2 + a_1 z f_0 + a_1 z f_1 z + a_1 z f_2 z^2 + a_2 z^2 f_0 + a_2 z^2 f_1 z + a_2 z^2 f_2 z^2; \end{aligned} \tag{2.8}$$

then, collecting terms of like powers of z gives

$$a(z)f(z) = a_0 f_0 + \left[a_0 f_1 + a_1 f_0\right] z + \left[a_0 f_2 + a_1 f_1 + a_2 f_0\right] z^2 + \left[a_1 f_2 + a_2 f_1\right] z^3 + \left[a_2 f_2\right] z^4. \tag{2.9}$$

Examination of this result shows that the coefficient of each power of z is composed of all combinations of the coefficients of $a(z)$ and $f(z)$ whose subscripts sum to the value of the exponent of z. In fact, the general pattern for the coefficient of z^j is $\sum_k a_k f_{j-k}$ where the sum extends over all relevant values. Comparison with equation (2.6) allows the conclusion: *the product of the z transforms of two signals gives the z transform of the convolution of those signals.*

These concepts establish a link between the theory of discrete, digital filters and the algebra of polynomials. This link has far reaching implications in signal theory because it allows the many properties of polynomials to be used in signal processing. For example, if polynomial multiplication can perform a convolution, then polynomial division does *deconvolution.* Later in this chapter, it will be demonstrated that the z transform is a generalization of the *discrete Fourier transform* provided that z is a complex variable. This means that the result demonstrated above is an example of the *convolution theorem* that will be proven for Fourier transforms in a later section. It will also be demonstrated that the location of the *roots or zeros* of $a(z)$ completely determines the behavior of $\{a\}$ as a digital filter.

Exercise 2.1.4. *Use z transforms to repeat the convolution of exercise 2.1.1.*

Exercise 2.1.5. *Given a sequence $\{a\}$, show that $a(z)z^m$ is the spectrum of the sequence delayed by m samples. Thus z is the unit delay operator. What is the meaning of z^{-1}?*

2.1.3 Convolution as a matrix multiplication

Usually the integral of the product of two continuous functions can be approximated in a computer as a matrix-vector product. The convolution integral provides an important example. Consider the discrete convolution formula

$$g_j = \sum_{k=0}^{N-1} a_{j-k} f_k \tag{2.10}$$

and compare it to the general formula for the product of a matrix, M_{jk} with a vector v_k that is

$$u_j = \sum_k M_{jk} v_k. \tag{2.11}$$

Examination of equations (2.6) and (2.11) shows that convolution can be done by defining a matrix $A_{jk} = a_{j-k}$. Written as a matrix operation, equation (2.10) becomes

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \dots & a_{-N+1} \\ a_1 & a_0 & a_{-1} & \dots & a_{-N+2} \\ a_2 & a_1 & a_0 & \dots & a_{-N+3} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N-1} & a_{N-2} & a_{N-3} & \dots & a_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{bmatrix}. \tag{2.12}$$

(If $\{a\}$ is causal, then the entries above the main diagonal of the matrix will all be zero.) This will be written as an abstract matrix equation as

$$\underline{g} = \underline{\underline{A}}\,\underline{f} \tag{2.13}$$

where the convention is that single underlined quantities are column vectors and a double underline denotes a matrix.

The matrix $\underline{\underline{A}}$ in equation (2.12) has a special symmetry that is required to perform a convolution. Though the matrix has N^2 entries, there are only N independent numbers that all come from $\{a\}$. The elements along any diagonal are constant. Each column contains a copy of $\{a\}$ that has been shifted to place a_0 on the main diagonal. Equivalently, each row contains a time-reversed and shifted copy of $\{a\}$. Matrices with this symmetry are called *Toeplitz* or sometimes *convolution* matrices. Since convolution is commutative, it follows that forming a Toeplitz matrix with $\{f\}$ and a column vector with $\{a\}$ must give an identical result.

One way to visualize the matrix-vector multiplication of equation (2.12) is called *matrix multiplication by rows*. In this process, each row of $\underline{\underline{A}}$ multiplies $\underline{f}$ as a vector dot product. That is, the corresponding elements are multiplied and the resulting products summed to a scalar that becomes the corresponding element of $\underline{g}$. This can be written as the set of equations

$$\begin{aligned} g_0 &= a_0 f_0 + a_{-1} f_1 + a_{-2} f_2 + \ldots a_{-N+1} f_{N-1} \\ g_1 &= a_1 f_0 + a_0 f_1 + a_{-1} f_2 + \ldots a_{-N+2} f_{N-1} \\ g_2 &= a_2 f_0 + a_1 f_1 + a_0 f_2 + \ldots a_{-N+3} f_{N-1} \\ \ldots &= \ldots \\ g_{N-1} &= a_{N-1} f_0 + a_{N-2} f_1 + a_{N-3} f_2 + \ldots a_0 f_{N-1}. \end{aligned} \tag{2.14}$$

This view of matrix multiplication shows how each sample of $\underline{g}$ (each output sample) is created as a linear superposition of the inputs. An alternate, but equally valid perspective comes from *matrix multiplication by columns*. Inspection of the equations (2.14) shows that each sample of $\underline{f}$ multiplies a corresponding column of $\underline{\underline{A}}$. Thus, a column oriented view of matrix multiplication suggests

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix} f_0 + \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ \vdots \\ a_{N-2} \end{bmatrix} f_1 + \begin{bmatrix} a_{-2} \\ a_{-1} \\ a_0 \\ \vdots \\ a_{N-3} \end{bmatrix} f_2 + \ldots \begin{bmatrix} a_{-N+1} \\ a_{-N+2} \\ a_{-N+3} \\ \vdots \\ a_0 \end{bmatrix} f_{N-1} \tag{2.15}$$

In this view, the formation of a convolution by a scaled superposition of impulse responses (the columns of $\underline{\underline{A}}$) is manifest. The final result is obtained for all samples simultaneously after the superposition of the final scaled column of $\underline{\underline{A}}$.

2.1.4 Convolution as a weighted average

Equation (2.15) shows that matrix multiplication by columns is equivalent to viewing convolution as the scaled superposition of impulse responses. However, equation (2.14) suggests an alternative, and equally valid, interpretation of convolution. This equation shows each element of $\underline{g}$ being formed as a weighted average of the samples of $\underline{f}$ with $\underline{\underline{A}}$ providing the weights. Of course, since convolution is commutative, $\underline{g}$ can equally well be regarded as a weighted average of $\underline{a}$ with a Toeplitz matrix $\underline{\underline{F}}$ providing the weights.

This view of convolution is valuable since it is often desirable to smooth a physical dataset. For example, a time series that represents the measurement of a slowly changing physical quantity may fluctuate more rapidly than is easily explicable. Such fluctuations may be ascribed to contamination of the measurements by random noise. In this case, the data are often subjected to a *running average* whereby the k^{th} output point is formed as the average of the input points on either side of sample k. This is also called *smoothing*. The *averaging width*, or number of points involved in the average,

is an important decision. For a width of m, a running average can be implemented as a convolution with a series of m ones, or an m-length boxcar. For example, a 3-length running average uses an averaging operator defined by $a_k = 1/3, -1 \leq k \leq 1; a_k = 0, otherwise$. Then, $\underline{f}$ is smoothed by the matrix equation

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \\ \vdots \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 1 & 0 & \dots & 0 \\ & & \dots & \dots & \dots & & \\ & & & \dots & \dots & \dots & \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ \vdots \end{bmatrix}. \tag{2.16}$$

Here the convolution matrix is nonzero only in the main diagonal and the two sub-diagonals above and below the main. These diagonals are filled with ones. Comparing with equation (2.14) shows that, for example, $g_2 = (f_1 + f_2 + f_3)/3$. Thus the conclusion is that a running average is equivalent to convolution with a boxcar.

It is easily concluded that a running average using weights other than unity can be expressed as convolution with an appropriate averaging function. For example, weights that ramp linearly from zero to one and back zero can be expressed as a triangular averaging function. Multi-dimensional averaging works the same way, it can always be expressed as a multi-dimensional convolution. Later in this chapter, the connection between averaging and *lowpass filtering* will be illustrated.

2.2 The Fourier Transform

The development of the Fourier transform is one of the most significant developments in mathematical analysis. Its digital counterpart, the discrete Fourier transform, plays a central role in signal analysis. Fourier developed his ideas in the context of solving the heat equation to study heat dissipation while boring out canons. Despite this specific intent, it soon became apparent that Fourier's theory could be used to develop solutions to other partial differential equations such as Laplace's equation or the wave equation. Despite the success of practical applications, the mathematics community resisted Fourier's ideas because of the difficult challenges they posed for analysis. For example, a discontinuous function, such as the boxcar function, could be synthesized from the superposition of an appropriate set of sine and cosines that are everywhere continuous. Also problematic was the convergence of Fourier's series of sines and cosines. The resolution of these and other similar questions is a fascinating chapter in the history of mathematics but is beyond the scope of this text.

2.2.1 The temporal Fourier transform and its inverse

In essence, Fourier's theory provides a means to represent an arbitrary function as a superposition (sum or integral) of a set of simpler functions called *basis functions*. Usually, these basis functions are the trigonometric sines and cosines of different frequencies. To construct a specific function, the sines and cosines must have amplitudes and phases that depend on frequency. Considered as a function of frequency, the amplitudes comprise the *amplitude spectrum* and the phases are the *phase spectrum* of the function being represented.

Fourier's theory can be defined on either a finite interval or an infinite domain. On a finite interval, only a finite number of basis functions are required. Their superposition is a summation over a countable set of frequencies and the method is called Fourier series. More relevant to data processing is the case of an infinite interval. Here an infinite number of basis functions are required

and their superposition is accomplished by an integration over frequency. This is called the *Fourier transform*. Rather than treating $\sin(\omega t)$ and $\cos(\omega t)$ as separate basis functions, it is common to use $\exp(i\omega t) = e^{i\omega t}$. Since Euler's theorem says $e^{i\omega t} = \cos(\omega t) + i\sin(\omega t)$ and since real and imaginary parts of an equation are independent, the complex exponential provides a decomposition equivalent to independent sin and cosine decompositions.

An essential tool in Fourier theory is an *orthogonality relation*. For complex exponentials, this takes the form

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i(\omega-\hat{\omega})t}dt = \delta(\omega-\hat{\omega}) \tag{2.17}$$

where $\delta(\omega-\hat{\omega})$ is Dirac's delta function that is defined by

$$\int_{-\infty}^{\infty}\delta(x)dx = 1 \tag{2.18}$$

$$\delta(x) = 0, x \neq 0 \tag{2.19}$$

and has the fundamental property that

$$\int_{-\epsilon}^{\epsilon} s(x)\delta(x)dx = s(0) \tag{2.20}$$

where $\epsilon > 0$ and $s(x)$ is any function. This result is called the *sifting property* of the Dirac's delta function.

The delta function is actually not a function in the classical sense because it has vanishing support and takes on a singular value (infinity) where the argument vanishes. It has been given rigorous meaning through the branch of analysis known as distribution theory. There it is modelled as the limit of a sequence of ordinary functions. These functions must have unit area but the limit is taken as their support vanishes. For example, a boxcar of width a and height a^{-1} has unit area and converges to a delta function in the limit as $a \to 0$.

A proof of equation (2.17), that does not assume the Fourier inversion theorem, requires distribution theory and that is beyond the scope of this discussion. Intuitively, it can be understood quite simply. If $\omega = \hat{\omega}$, then $\exp(\omega t - \hat{\omega}t) = 1$ and the integral in equation (2.17) is infinite. On the other hand if $\omega \neq \hat{\omega}$, then $\exp(\omega t - \hat{\omega}t) = \cos((\omega-\hat{\omega})t) + i\sin((\omega-\hat{\omega})t)$. Both terms are simply periodic oscillatory functions that, over the domain $-\infty < t < \infty$ will integrate to zero.

Given the orthogonality relation, the Fourier transform theorem can be derived by postulating an expansion of an arbitrary function $s(t)$ in terms of the complex exponentials

$$s(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega)e^{-i\omega t}d\omega. \tag{2.21}$$

This says that a representation of $s(t)$ is sought as the superposition of an infinite set of complex exponentials having weights $S(\omega)$. At this point, $S(\omega)$ is unknown but it can be calculated by multiplying equation (2.21) by $\exp(i\hat{\omega}t)$ and integrating over t, that is

$$\int_{-\infty}^{\infty} s(t)e^{i\hat{\omega}t}dt = \int_{-\infty}^{\infty} e^{i\hat{\omega}t}\left[\frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega)e^{-i\omega t}d\omega\right]dt. \tag{2.22}$$

Now, interchange the order of integration of the right and use the orthogonality relation, equation

(2.17),

$$\int_{-\infty}^{\infty} s(t)e^{i\hat{\omega}t}dt = \int_{-\infty}^{\infty} S(\omega)\left[\frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\hat{\omega}t-i\omega t}dt\right]d\omega = \int_{-\infty}^{\infty} S(\omega)\delta(\hat{\omega}-\omega)d\omega = S(\hat{\omega}). \tag{2.23}$$

Upon renaming $\hat{\omega}$ to ω, this is just $S(\omega)$, which will be called the *Fourier transform* of $s(t)$. In summary, the (forward) Fourier transform of $s(t)$ is given by

$$S(\omega) = \int_{-\infty}^{\infty} s(t)e^{i\omega t}dt \tag{2.24}$$

and the inverse Fourier transform is

$$s(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega)e^{-i\omega t}d\omega. \tag{2.25}$$

If a function is denoted by a lower case letter, then its Fourier transform is denoted by a capital letter. In geophysics, it is often desirable to use the frequency variable, f, in units of *Hertz* or cycles-per-second. The variable ω, called *angular frequency* in equations (2.24) and (2.25) is measured in radians-per-second. Since there are 2π radians in one cycle of a sine wave, these variables are related by $\omega = 2\pi f$. Upon change of variables, the Fourier transform pair in *cyclical frequency* is

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{i2\pi ft}dt \tag{2.26}$$

and

$$s(t) = \int_{-\infty}^{\infty} S(f)e^{-i2\pi ft}df. \tag{2.27}$$

In addition to the selection of frequency variables, there are other arbitrary choices in the definition of a Fourier transform pair. Things work equally well if the sign in the *Fourier kernal* (i.e. exponent of the complex exponential) is negative for the forward transform and positive for the inverse transform. Also, the $(2\pi)^{-1}$ factor in front of the inverse transform (equation (2.25)) can be replaced by $(2\pi)^{-1/2}$ in front of both transforms. These arbitrary choices appear in all possible combinations throughout the relevant literature. The practitioner must be aware of this and always check the definitions that are in effect before using a particular theorem or result. In this book, the temporal Fourier transforms will always be written as in the equations (2.24)-(2.27); however, the choice of angular or cyclical frequency variables may vary as is convenient.

2.2.2 Real and imaginary spectra, even and odd functions

For a real or complex-valued function, $s(t)$, the Fourier transform is always complex-valued. It is customary to call $S(f)$ either the Fourier transform or the *spectrum*. These terms will be used synonymously in this text. As a complex-valued function, the spectrum has both real and imaginary parts that can be denoted in either of two equivalent ways

$$\begin{aligned} S(f) &= S_R(f) + iS_I(f) \\ S(f) &= Re(S(f)) + iIm(S(f)). \end{aligned} \tag{2.28}$$

An alternate decomposition of the spectrum is employed more commonly than real and imaginary parts. The *amplitude spectrum*, $A_S(f)$, and the *phase spectrum*, $\phi_S(f)$, are defined as

$$A_S(f) = \sqrt{S_R^2(f) + S_I^2(f)} \tag{2.29}$$

and

$$\phi_S(f) = \arctan(\frac{S_I(f)}{S_R(f)}). \tag{2.30}$$

Thus the spectrum can be written

$$S(f) = A_S(f)e^{i\phi_S(f)} \tag{2.31}$$

. Equations (2.28) and (2.31) are equivalent and alternate representations of a spectrum. Either one can be used as convenient for whatever problem is at hand.

The Fourier transform decomposes a function using the complex exponential, $\exp(i\omega t)$ as a basis. By Euler's theorem, the is equivalent to $\cos\omega t + i\sin\omega t$. The cosine functions all are *even* functions of t while the sines are *odd* functions. An even function has the property that $s(-t) = s(t)$ while an odd function has the property that $s(-t) = -s(t)$. Any function can be expressed as a sum of even and odd parts.

$$s(t) = s_e(t) + s_o(t) \tag{2.32}$$

where

$$s_e(t) = \frac{1}{2}\left[s(t) + s(-t)\right] \tag{2.33}$$

and

$$s_e(t) = \frac{1}{2}\left[s(t) - s(-t)\right]. \tag{2.34}$$

Even functions have the property that $\int_{-a}^{a} s_e = 2\int_0^a s_e$ while for odd functions $\int_{-a}^{a} s_0 = 0$. Furthermore, the product of two odd functions, or the product of two even functions, is even while an even function times an odd function is odd. From these properties it follows directly that

$$\int_{-\infty}^{\infty} s_e(t)e^{i\omega t}dt = Re\left(\int_{-\infty}^{\infty} s(t)e^{i\omega t}dt\right) = Re\left(S(\omega)\right). \tag{2.35}$$

That is, the real part of the spectrum of $s(t)$ is the Fourier transform of the even part of $s(t)$. Similarly, it is also true that

$$\int_{-\infty}^{\infty} s_o(t)e^{i\omega t}dt = Im\left(\int_{-\infty}^{\infty} s(t)e^{i\omega t}dt\right) = Im\left(S(\omega)\right). \tag{2.36}$$

This means that the imaginary part of the spectrum of $s(t)$ is the spectrum of the odd part of $s(t)$.

Exercise 2.2.1. *Show that the product of two odd functions or of two even functions is even.*

Exercise 2.2.2. *Show that* $\int_{-a}^{a} s_o(t)dt = 0$ *where* a *is a positive constant and* s_o *is any odd function.*

Exercise 2.2.3. *Prove equations* (2.35) *and* (2.36).

2.2.3 The convolution theorem and the differentiation theorem

A very important property of the Fourier transform is that it converts convolution into multiplication. That is, if $s(t)$ is the convolution of $u(t)$ and $v(t)$, the spectrum of $s(t)$ of equation (2.3) is the product

of the spectra of $u(t)$ and $v(t)$. To prove this, substitute into equation (2.3) the definitions of $u(t)$ and $v(t)$ as inverse Fourier transforms of their spectra. This gives

$$s(t) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} U(f) e^{-i2\pi f(t-\tau)} df \right] \left[\int_{-\infty}^{\infty} V(\hat{f}) e^{-i2\pi \hat{f}\tau} d\hat{f} \right] d\tau. \tag{2.37}$$

Now, move the τ integral to the inside

$$s(t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U(f) V(\hat{f}) e^{-i2\pi ft} \left[\int_{-\infty}^{\infty} e^{-i2\pi(\hat{f}-f)\tau} d\tau \right] df d\hat{f}. \tag{2.38}$$

The quantity in square brackets is $\delta(\hat{f} - f)$ and this then collapses the $\hat{f}$ integral to give

$$s(t) = \int_{-\infty}^{\infty} U(f) V(f) e^{-i2\pi ft} df. \tag{2.39}$$

Recognizing this integral as an inverse Fourier transform proves the basic result that $S(f) = U(f)V(f)$ and this is called *the convolution theorem*.

The mathematics shown in equations (2.37)-(2.38) requires considerable effort to justify with full mathematical precision. There are subtle but difficult questions such as: "For what class of functions do the various integrals converge?" and "Under what circumstances may the order of integration be changed?". These questions have been answered and the operations are valid for a very general class of functions and even distributions. The interested reader should consult a treatise in *harmonic analysis* such as Korner (1988) or Stein (1993). For this book, it will be assumed that the convolution theorem is valid for all functions encountered.

The importance of this result is that it means that a filter can be applied to a signal in the Fourier domain by multiplying the spectrum of the signal by that of the filter. For digital computations, this is often faster than direct evaluation of the convolution integral because of the *fast Fourier transform* algorithm. However, there is more to this story that will be discussed after the *discrete Fourier transform* is introduced.

The symmetry of the Fourier transform means that the time and frequency domains are essentially similar. Thus is follows the the convolution theorem also works in reverse. That is, the product $u(t)v(t)$ is the convolution of the spectra of these functions in the frequency domain. A careful derivation shows that a factor of $(2\pi)^{-1}$ with the result

$$\int_{-\infty}^{\infty} u(t)v(t)e^{i\omega t} dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} U(\hat{\omega}) V(\omega - \hat{\omega}) d\hat{\omega} \tag{2.40}$$

Also of great importance is that the Fourier transform reduces differentiation to multiplication. For example, consider the calculation of $\partial_t s(t)$ given $s(t)$ as an inverse Fourier transform of its spectrum

$$\frac{\partial s(t)}{\partial t} = \frac{\partial}{\partial t} \int_{-\infty}^{\infty} S(f) e^{-i2\pi ft} df. \tag{2.41}$$

Examination of the right-hand-side of this expression shows that the t dependence occurs only in the exponent of the complex exponential. Therefore the differentiation can be moved under the integration sign and performed analytically with the result

$$\frac{\partial s(t)}{\partial t} = \int_{-\infty}^{\infty} -i2\pi f S(f) e^{-i2\pi ft} df. \tag{2.42}$$

Thus differentiation with respect to t can be performed by multiplying the spectrum of $s(t)$ by $-i2\pi f = -i\omega$. In this case, angular frequency leads to a simple formula.

More general differentiation operators can also be computed. Let $a(t, \partial_t) = b_1(t)\partial_t + b_2(t)\partial_t^2$ be a differential operator, then

$$\begin{aligned} a(t, \partial_t)s(t) &= b_1(t)\frac{\partial s(t)}{\partial t} + b_2(t)\frac{\partial^2 s(t)}{\partial t^2} = \frac{1}{2\pi}\int_{-\infty}^{\infty}\left[-i\omega b_1(t) - \omega^2 b_2(t)\right] S(\omega)e^{-i\omega t}d\omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\alpha(t,\omega)S(\omega)e^{-i\omega t}d\omega \end{aligned} \tag{2.43}$$

where $\alpha(t, \omega) = -i\omega b_1(t) - \omega^2 b_2(t)$ is called the *symbol* of the differential operator. In this manner virtually any differential operator can be converted into an equivalent Fourier-domain multiplicative operator. This observation means that the Fourier transform is of great importance in solving partial differential equations.

Exercise 2.2.4. *Derive equation* (2.40).

2.2.4 The phase-shift theorem

Another important result involving Fourier transforms is the relationship between a time shift and a phase shift. Consider the Fourier transform of the time-shifted signal $s(t + \Delta t)$

$$\hat{S}(f) = \int_{-\infty}^{\infty} s(t + \Delta t)e^{i2\pi ft}dt \tag{2.44}$$

and make the change of variables $\tau = t + \Delta t$ so that

$$\hat{S}(f) = \int_{-\infty}^{\infty} s(\tau)e^{i2\pi f(\tau-\Delta t)}d\tau = e^{-i2\pi f\Delta t}\int_{-\infty}^{\infty} s(\tau)e^{i2\pi f\tau}d\tau = e^{-i2\pi f\Delta t}S(f). \tag{2.45}$$

In the final expression, $S(f)$ is the Fourier transform of $s(t)$. This shows that the spectrum of the time-shifted signal may be computed from the spectrum of the original signal by multiplication with $e^{-i2\pi f\Delta t}$. This operator has unit amplitude and a phase that is linear in f with a slope $-2\pi f\Delta t = -\omega\Delta t$. Let $S(f) = A_S(f)e^{i\phi_S(f)}$ and equation (2.45) becomes

$$\hat{S}(f) = S(f)e^{-i2\pi f\Delta t} = A_S(f)e^{i\phi_S(f)}e^{-i2\pi f\Delta t} = A_S(f)e^{i\phi_S(f)-i2\pi f\Delta t}. \tag{2.46}$$

Thus, the spectrum of the time-shifted signal is obtained from the spectrum of the original signal by a *phase shift* . That is, a linear term is added to the phase. The magnitude of the slope of the linear term is directly proportional to the magnitude of the time shift.

One subtlety is that the overall sign on the phase-shift depends upon the sign convention used in the Fourier transform definition. That is, equation (2.46) results from the Fourier transform pair of equations (2.26)-(2.27). If the sign convention is reversed so that the forward transform is done with $exp{-}i2\pi ft$, then the sign of the phase shift required for a positive time shift will be reversed. Thus, if a theory is derived analytically with one sign convention and then implemented with software that uses the other sign convention, confusing results can ensue. As a practical rule, if a phase-shift program does not give the expected results, then it should be tested with the sign reversed on the phase.

The phase-shift theorem is especially important with sampled data. This is because a time-shift that is not an integral number of samples required that new samples be interpolated from the signal.

If the time shift is done as a phase shift, then this interpolation is not required. In a sense, it is achieved implicitly by the discrete Fourier transform.

2.2.5 The spectrum of a real-valued signal

Fourier transform theory is developed such that both the time-domain signal and its spectrum can be complex valued. This leads to relationships between the real and imaginary parts in both domains. On the other hand, data is inherently real valued but its spectrum is still complex. Considering that the real and imaginary parts of a general complex function are independent, this means that the spectrum of a real signal has two functions that are calculated from one. This is only possible if there is a relationship between the real and imaginary parts of the spectrum of a real signal. Put another way, the spectrum must have a *symmetry*.

Consider a real signal written as the inverse Fourier transform of its spectrum

$$s(t) = \int_{-\infty}^{\infty} S(f) e^{-i2\pi ft} df. \tag{2.47}$$

Since a real signal is equal to its complex conjugate[1], then it must be true that

$$\int_{-\infty}^{\infty} S(f) e^{-i2\pi ft} df = \int_{-\infty}^{\infty} \bar{S}(f) e^{i2\pi ft} df \tag{2.48}$$

where $\bar{S}$ denotes the complex conjugate of S. The complex conjugation has changed the right-hand-side from an inverse to a forward Fourier transform. Substitution of $\hat{f} = -f$ will change it back to an inverse transform. This gives

$$\int_{-\infty}^{\infty} S(f) e^{-i2\pi ft} df = \int_{\infty}^{-\infty} \bar{S}(-\hat{f}) e^{-i2\pi \hat{f}t} d(-hatf) = \int_{-\infty}^{\infty} \bar{S}(-\hat{f}) e^{-i2\pi \hat{f}t} d\hat{f}. \tag{2.49}$$

In the final integral, $\hat{f}$ can be freely renamed to f because in a definite integral the name of the variable of integration is irrelevant. If this is surprising, consider that $\int_0^4 xdx, \int_0^4 ydy,$ and $\int_0^4 \eta d\eta$ are all equal to the same number, 8. So, equation (2.49) shows that the spectrum of a real-valued signal has the symmetry

$$\begin{gathered} S(f) = \bar{S}(-f) \\ \text{or} \\ \bar{S}(f) = S(-f). \end{gathered} \tag{2.50}$$

This means that the negative frequencies are not independent of the positive ones. Rather, one may always be calculated from the other.

This symmetry has importance in data processing because it means that only half of the Fourier transform need be computed, stored, and operated upon.

Exercise 2.2.5. *Show that the amplitude spectrum of a causal, real function is an even function and that the phase spectrum is an odd function.*

[1]The complex conjugate of a complex number z is written as $\bar{z}$ and is defined as $\bar{z} = Rez - iImz$. The complex conjugate may also be formed by replacing i by $-i$ wherever it occurs in a complex-valued expression.

2.2.6 The spectrum of a causal signal

In the previous section, it was demonstrated that a real-valued signal imposes a certain symmetry upon its spectrum. It is a similar story with a causal signal, that is, a function, s(t), for which $s(t) = 0, t < 0$. This symmetry can be derived from the time-domain requirement for a causal function that

$$s(t) = h(t)s(t) \tag{2.51}$$

where $h(t)$ is the *Heaviside step function* or *unit causal function* defined by

$$h(t) = \begin{cases} 1, t \geq 0 \\ 0, \text{otherwise} \end{cases} . \tag{2.52}$$

Since, equation (2.51) is a product of two temporal functions, the analogous frequency-domain equation must involve a convolution of two spectra. The development of this expression will lead to the desired symmetry.

This requires the Fourier transform of the step function, and that is known to be

$$H(\omega) = \pi\delta(\omega) + \frac{i}{\omega}. \tag{2.53}$$

An informal justification of this result begins with fact that the derivative of the step function is the delta function

$$\frac{\partial h(t)}{\partial t} = \delta(t). \tag{2.54}$$

Thus, if $\delta(t)$ were to be integrated, the step function would be recovered to within a possible additive constant. In section 2.2.3 it was shown that differentiation in the time domain corresponds to multiplication by $-i\omega$ in the frequency domain. It follows that integration must correspond to $(-i\omega)^{-1} = i/\omega$. Since the Fourier transform of $\delta(t)$ is the constant 1, it might be expected that the Fourier transform of $h(t)$ is i/ω. However, a moments reflection shows that this cannot be the entire story since a purely imaginary spectrum corresponds to an odd function in the time domain (see section 2.2.2) and $h(t)$ is not purely odd. What is lacking from this result is the Fourier transform of the even part of $h(t)$. Using equation (2.33), the even part of $h(t)$ is just the constant 1/2. (What is the odd part?) Since the Fourier transform of 1/2 is $\pi\delta(\omega)$, the spectrum of $h(t)$ must be $\pi\delta(\omega)+i/\omega$ and this is just equation (2.53). A more formal proof can be found in Lancaster and Salkauskas (1996).

It follows from the convolution theorem and equations (2.53) and (2.40) that the Fourier transform of equation (2.51) is

$$S(\omega) = \frac{1}{2\pi}S(\omega) \bullet \left[\pi\delta(\omega) + \frac{i}{\omega}\right] \tag{2.55}$$

where $\bullet$ denotes a convolution over ω. Since $S(\omega) \bullet \delta(\omega) = S(\omega)$, this becomes

$$S(\omega) = \frac{i}{\pi}S(\omega) \bullet \frac{1}{\omega}. \tag{2.56}$$

The factor of i on the right-hand-side essentially switches the real and imaginary parts around. Decomposing equation (2.56) into its real and imaginary parts gives

$$S_R(\omega) = \frac{-1}{\pi}S_I(\omega) \bullet \frac{1}{\omega} = \frac{-1}{\pi}\int_{-\infty}^{\infty} \frac{S_I(\hat{\omega})}{\omega - \hat{\omega}} d\hat{\omega} \tag{2.57}$$

and

$$S_I(\omega) = \frac{1}{\pi} S_R(\omega) \bullet \frac{1}{\omega} = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{S_R(\hat{\omega})}{\omega - \hat{\omega}} d\hat{\omega}. \tag{2.58}$$

The integrals on the right-hand-sides of these equations are explicit prescriptions for convolution with ω^{-1} and are called *Hilbert transforms*. Formally, the Hilbert transform of a function $u(x)$ is defined as

$$H\left[u\right](x) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{u(\hat{x})}{x - \hat{x}} d\hat{x} \tag{2.59}$$

so that equations (2.57) and (2.58) can be written compactly as

$$S_R(\omega) = -H\left[S_I\right](\omega) \tag{2.60}$$

and

$$S_I(\omega) = H\left[S_R\right](\omega). \tag{2.61}$$

These are the final results of this section. A causal signal has the symmetry that the real and imaginary parts of its spectrum are not independent but rather are Hilbert transforms of one another.

2.2.7 Minimum phase

Consider a physical signal, $s(t)$ with the properties that it has finite energy and that an inverse filter, also with finite energy, can be constructed for it. The finite energy requirement for $s(t)$ can be written

$$\text{total energy} = 2 \int_0^{\infty} A_S^2(\omega) d\omega < \infty. \tag{2.62}$$

The inverse filter, $\hat{s}(t)$, is defined by $\hat{s}(t) \bullet s(t) = \delta(t)$ from which it is clear that its amplitude spectrum must be $A_S^{-1}(\omega)$. So the energy for the inverse filter must be

$$\text{total energy of inverse} = 2 \int_0^{\infty} A_S^{-2}(\omega) d\omega < \infty. \tag{2.63}$$

It is difficult, if not impossible, to satisfy both of these equations simultaneously. For example, suppose $A_S(\omega) > 0$ for all finite ω, then equation (2.62) is only satisfied if $A_S(\omega) \to 0$ rapidly as $\omega \to \infty$. However, this condition means that $A_S^{-1}(\omega) \to \infty$ and so equation (2.63) will diverge. Therefore, it seems prudent to relax the conditions demanded of $s(t)$ while still maintaining sufficient generality to model all physical systems. It will be assumed that, for a causal $s(t)$, an ω_{max} can be found such that

$$A_S(\omega) \to \begin{cases} > 0, |\omega| < \omega_{max} \\ = 0, \omega \geq \omega_{max} \end{cases} . \tag{2.64}$$

This is a very reasonable requirement for any physical system though it does exclude some interesting mathematical signals. Obviously, the inverse filter cannot invert this amplitude spectrum for $\omega \geq \omega_{max}$ so a *bandlimited* inverse must be acceptable. This is defined through

$$A_{\hat{S}}(\omega) = \begin{cases} \frac{1}{A_S(\omega)}, |\omega| < \omega_{max} \\ 0, \omega \geq \omega_{max} \end{cases} . \tag{2.65}$$

Signals with finite energy such as those defined by equations (2.64) and (2.65) are called *stable* signals.

Now consider the signal defined by the spectrum

$$S_{log}(\omega) = \begin{cases} \ln S(\omega) = \ln A_S(\omega) + i\phi_S(\omega), \omega < \omega_{max} \\ 0, \omega \geq \omega_{max} \end{cases} . \tag{2.66}$$

If it can be shown that $S_{log}(\omega)$ corresponds to a causal signal in the time domain, then the results of section 2.2.6 apply and it can be concluded that $\ln A_S = -H[\phi_S]$ and $\phi_S = H[\ln A_S]$. The inverse Fourier transform of equation (2.66) is

$$s_{log}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_{log}(\omega)e^{-i\omega t} = \frac{1}{2\pi}\int_{-\omega_{max}}^{\omega_{max}} [\ln A_S(\omega) + i\phi_S(\omega)]\, e^{-i\omega t}d\omega. \tag{2.67}$$

Equation (2.67) can be considered to be part of a contour integral in the complex ω plane. In addition to the real ω axis from $-\infty$ to ∞ let the contour be closed by a semi-circle of radius tending to ∞ either above the real axis, $Im(\omega) > 0$, or below, $Im(\omega) < 0$. For complex ω, the complex exponential becomes $\exp(i\omega_R t - \omega_I t)$, so for positive t the contour is completed above the real axis in order that the real exponential is decaying. Similarly, for negative t, the contour is completed below the real axis. Call contour along the real axis C1, that around the lower semi-circle C2, and that around the upper semi-circle C3.

Cauchy's integral theorem says that a contour integral of a function of a complex variable around a closed contour is always zero if the function is everywhere *analytic* within the contour. An analytic function is one that is differentiable at every point. Thus, if S_{log} is analytic in the lower half-plane then its integral around C1+C2 is zero. Furthermore, if the integral along C2 vanishes, then so must that along C1. Since C1+C2 is the appropriate contour for $t < 0$, showing this would prove that s_{log} is a causal function.

By assumption, $S(\omega)$ is the spectrum of a causal function, therefore, the integral $\int_{-\infty}^{\infty} S(\omega)\exp i\omega t d\omega$ must be zero for $t < 0$. The integral along the contour C2 clearly vanishes because of the exponential attenuation caused by $\exp(-\omega_I t)$ for very large ω. Therefore, a necessary condition for $S(\omega)$ to be causal is that it is analytic in the lower half-plane. Now, does it follow that S_{log} is also analytic? This is true provided that $A_S(\omega)$ does not vanish for $\omega < \omega_{max}$ because the logarithm of 0 is $-\infty$. Since this condition was assumed for invertibility, it follows that S_{log} is analytic in the lower half-plane. The integral along C2 vanishes for the same reason as for $S(\omega)$.

Under the assumptions given previously, it follows that the spectrum defined by equation (2.66) is the spectrum of a causal function. Therefore, it follows that

$$Re\left(S_{log}\right)(\omega) = -H\left[Im\left(S_{log}\right)\right](\omega) \tag{2.68}$$

therefore from equation (2.66)

$$\ln\left[A_S(\omega)\right] = -H\left[\phi_S(\omega)\right] \tag{2.69}$$

and similar reasoning leads to the conclusion that

$$\phi_S(\omega) = H\left[\ln\left(A_S(\omega)\right)\right]. \tag{2.70}$$

In summary, for a causal, stable signal that has a causal, stable inverse, the log amplitude spectrum and the phase spectrum are a Hilbert transform pair. The phase spectrum defined by equation (2.70) is given the special name of *the minimum phase spectrum*. In a theorem credited to Robinson (XXXXXXXX) it is shown that, of all causal signals with the amplitude spectrum $A_s(\omega)$, the phase spectrum given by the Hilbert transform of the logarithm of the amplitude spectrum is the smallest

in absolute value at any frequency. In the time domain, the minimum phase signal is the most *front-loaded* signal possible that is both causal and has the given amplitude spectrum. If the phase is made smaller, then the signal becomes non causal.

Chapter 3

Wave propagation

3.1 Introduction

Seismic wave propagation in the upper layers of the earth's crust is a complex physical process. For example, a wave emanating from an explosive charge in a shallow (5 to 20 m) hole generally undergoes an immediate and rapid spectral decay before it leaves the near surface[1]. Theory predicts that this spectral decay is associated with *minimum phase* effects that produce the characteristic source waveform (Futterman, 1962). In addition to the primary downgoing pulse, there is always upgoing energy from the source that reflects off of the earth's free surface to produce a so-called *source ghost*. In fact, there will be an entire series of similar ghosts that are produced when either the primary or a particular ghost reflects off the base of the near surface and travels up again to reflect off the free surface. Thus, it must always be expected that the downgoing pulse that serves to illuminate the subsurface consists of a complex reverberatory series.

The subsurface is commonly idealized as a stacked sequence of nearly homogeneous layers, called *formations*, whose geometry may be simple or complex. The contacts between the formations are called *horizons* and are assumed to be in *welded* contact. (Of course, slip does occur along faults in the subsurface but this is negligible over the time-span of a seismic experiment.) In the case of a sedimentary basin, the formations are assumed to be horizontal and hence their material description depends only on the vertical coordinate. This book adopts the common convention of a right-handed Cartesian coordinate system with the z coordinate oriented in the vertical and increasing downward. Thus it is common to describe the sedimentary basin environment as a $v(z)$ setting by which it is meant that the propagation speed of seismic waves depends only upon the formation depth. (However, even in this setting, it is rarely a good model to assume that the base of the near surface is a horizontal boundary.)

More complex settings are found near mountain belts, along passive continental margins, and near the boundaries of sedimentary basins. Here, formations can be greatly distorted from the horizontal due to tectonic compressional or extensional forces. Common jargon for such settings is to say that they are $v(x, z)$ environments which means that seismic wave speed can depend arbitrarily upon position.

As a wave propagates into the subsurface, a number of important physical effects occur. When

[1]The phrase *near surface* refers to a deliberately vague region just below the earth's surface. It is generally considered to be a region of high attenuation where static delays occur. Near surface effects are expected to be *surface consistent*.

a wave encounters a horizon (with different material parameters on either side) both reflected and transmitted waves result. Regardless of the type of incident wave, reflections and transmissions are generally found of each possible type of waves called *compressional or P* , *shear or S*, and perhaps *interface* waves. Even if a source can be devised to produce only one type of wave, the propagating wavefield rapidly evolves to contain all possible types going in all possible directions. For idealized elastic media and plane waves [2] the equations governing reflection and transmission at a plane interface are known exactly. These algebraically complex equations are given by Aki and Richards (1980) and many approximate forms are also known.

Waves also lose energy as they propagate. Even in a perfectly elastic medium, a spherical wavefront suffers a $1/r$ amplitude loss that is a geometric consequence of spherical divergence (or spreading). In real media, there is always some additional loss due to anelasticity and typically quantified by the dimensionless *quality factor* Q. These anelastic effects are often modelled by the *constant* Q *theory* (Kjartansson, 1980) which refers to a Q that is independent of frequency but may vary with position. All anelastic loss theories predict that attenuation is necessarily accompanied by dispersion or they violate causality.

These introductory remarks only hint at the complexity of the true seismic wave propagation process in the earth's crust. They are mentioned here to give the reader some idea of the drastic simplifications that will be seen in the theoretical discussions in this chapter. Producing realistic numerical models of seismic wave propagation is a blend of both science and intuition. The science must be pushed to limits dictated by the available computation power but then approximations must be made. The science can be dealt with logically; but, knowing which physical effects are small and can be neglected or developing an approximate expression for a mathematical operator, is often a very subjective process.

This chapter will examine computational methods of wave propagation based mostly on the scalar wave equation. Some initial effort will be expended in justifying the mathematical form of the equation and understanding its range of validity. Then raytracing methods for simple media will be developed and used to illustrate fundamental effects. Next methods for the formal solution of the wave equation will be developed. The Fourier method will be shown to emerge from the classical technique of separation of variables. Then Green's theorem will be used to develop solutions which proceed by integrating over the boundary (or initial) values. Finally, finite difference methods will be explored to develop very general solution methods. (Note to the reader: this paragraph refers to the chapter as I hope it to be. It is far from that now. Sorry!)

3.2 The wave equation derived from physics

There are many different 'wave equations' that arise from physical theory (and many more that occur only in mathematics). The classical scalar wave equation, $\nabla^2\psi = v^{-2}\partial_t^2\psi$ (where ψ is the wave function, v is the wave propagation speed, and $\nabla^2\psi = \partial_x^2\psi + \partial_y^2\psi + \partial_z^2\psi$ is the Laplacian in 3D), is only one possibility. For example, pressure, P, in an inhomogeneous fluid will be seen to obey the wave equation $\vec{\nabla}\cdot(\rho(\mathbf{x})\vec{\nabla}P) = k(\mathbf{x})\partial_t^2 P$ where $\rho(\mathbf{x})$ and $k(\mathbf{x})$ are the heterogeneous density and bulk modulus and $\mathbf{x}$ is the position vector. Though apparently quite different, what these wave equation have in common with all other wave equations is that they are *hyperbolic partial differential equations* (pde's). A great body of literature exists concerning the solution of pde's and they are

[2] Plane waves are a mathematical idealization because they are necessarily infinite in extent. A point source actually emits spherical wavefronts which may be approximately planar if the radius of the wavefront is large. A spherical wavefront may be mathematically decomposed into plane waves for analysis.

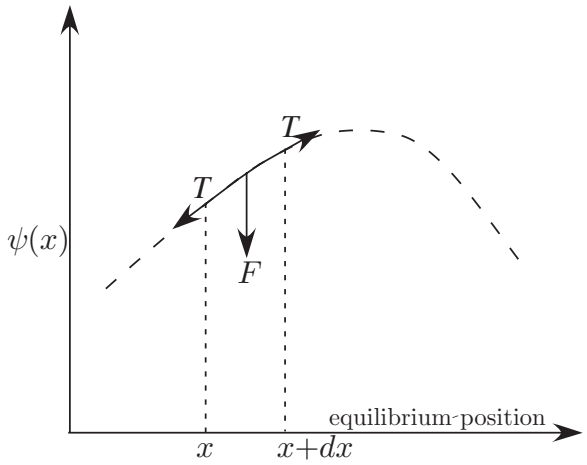

Figure 3.1: Tension, T acts on a vibrating string whose displacement is $\psi(x)$ to produce a restoring force F

known to fall into three general classes: *hyperbolic*, *parabolic*, and *elliptic*. A discussion of the origin of this classification and the meaning of these terms is beyond the scope of this book and the reader is invited to consult a more general reference. There are many excellent works and some that are particular favorites of this author are Morse and Feshbach (1953), Zauderer (1989), Ames (1992), and Durran (1999). These books are very practical in their orientation and cannot be said to present the subject of pde's with full mathematical rigor. For this purpose, the sophisticated reader may wish to consult Gustafson (1987) or, for the very brave, Taylor (1996).

In this section, some physical systems that lead to wave equations are examined. Only systems that are relevant to the seismic exploration problem will be discussed.

3.2.1 A vibrating string

Here we will examine the simplest second-order hyperbolic pde that arises as the governing equation for transverse waves on a vibrating string. The presentation is adapted from that found in Morse and Feshbach (1953). The solutions to this system are an instructive beginning for the more complex problems in two and three spatial dimensions. The one dimensional solutions can be considered as a basis for the common seismic processing step known as *stretching* that transforms a single seismic trace from time to depth or the reverse. Consider the situation shown in figure 3.1 where a snapshot (i.e. at constant time) of a vibrating string is shown. The displacement of the string from equilibrium, $\psi(x)$ is shown as a dashed curve and the tension, T, always acts tangentially to this curve. The tension on both ends of a differential element of the string from x to $x+dx$ creates a net force, F, that acts to restore the string to its equilibrium position. It is assumed that the magnitude of $\psi(x)$ is sufficiently small at all points such that there is no significant change in T or in the length of the string. At position x in Figure 3.1, $T(x)$ is directed tangentially along the string and has a component, $T(x)\sin\theta$, that acts to restore the string to it's equilibrium position. (Here, *theta* is the angle between the string and the horizontal.) The assumption that $T(x)$ is small implies that θ is

also small. This allows

$$T \sin\theta(x) \simeq T \tan\theta(x) = T\frac{\partial\psi(x)}{\partial x} \tag{3.1}$$

where the last step follows from the geometric definition of the derivative. Considering the tension acting at both ends of the string element between x and $x + dx$ allows the total restoring force on the element to be written as

$$F(x)dx = T\left[\frac{\partial\psi(x+dx)}{\partial x} - \frac{\partial\psi(x)}{\partial x}\right] \tag{3.2}$$

where $F(x)$ is the force per unit length and the minus sign in the square brackets is needed because the tension acts at x and $x + dx$ in opposite directions.

Now, recall that the definition of the second derivative of $\psi(x)$ is

$$\frac{\partial^2\psi(x)}{\partial x^2} = \lim_{dx\to 0}\frac{1}{dx}\left[\frac{\partial\psi(x+dx)}{\partial x} - \frac{\partial\psi(x)}{\partial x}\right]. \tag{3.3}$$

Thus in the limit as dx becomes infinitesimal, equation (3.2) leads to

$$F(x) = T\frac{\partial^2\psi(x)}{\partial x^2}. \tag{3.4}$$

This important result says that the restoring force (per unit length) depends on the local curvature of the string. When the second derivative is positive, the curvature is concave ($\smile$) and when it is negative the curvature is convex ($\frown$). Since a positive force is directed upward in Figure 3.1 and a negative force is downward, it is easy to see that this force does act to restore the string to equilibrium.

If the string's weight is negligible, then Newton's second law (force = mass × acceleration) gives

$$T\frac{\partial^2\psi(x,t)}{\partial x^2} = \mu\frac{\partial^2\psi(x,t)}{\partial t^2} \tag{3.5}$$

where μ is the mass per unit length and the dependence of ψ on both space and time is explicitly acknowledged. It is customary to rearrange equation (3.5) to give the standard form for the *one-dimensional scalar wave equation* as

$$\frac{\partial^2\psi(x,t)}{\partial x^2} - \frac{1}{v^2}\frac{\partial^2\psi(x,t)}{\partial t^2} = 0 \tag{3.6}$$

where $v = \sqrt{\frac{T}{\mu}}$ is the *wave velocity*[3] This analysis has shown that the scalar wave equation arises for the vibrating string as a direct consequence of Newton's second law. Wave equations invariably arise in this context, that is when a displacement is initiated in a continuum. Also typical is the assumption of small displacements. Generally, large displacements lead to nonlinear equations. The waves modelled here are known as *transverse* waves because the particle displacement is in a directional orthogonal to the direction of wave propagation. In section **??** *longitudinal* waves, that have particle oscillation in the direction of wave propagation, are discussed.

If there are external forces being applied to the string as specified by the *force density*[4] function

[3]It is quite common to use *velocity* for this scalar quantity even though velocity is classically a vector quantity in physics. This book conforms with this common usage.

$S(x,t)$, then equation (3.6) is usually modified to

$$\frac{\partial^2\psi(x,t)}{\partial x^2} - \frac{1}{v^2}\frac{\partial^2\psi(x,t)}{\partial t^2} = \frac{S(x,t)}{T}. \tag{3.7}$$

Both equations (3.6) and (3.7) are examples of one-dimensional hyperbolic pde's with constant coefficients. Equation (3.6) is said to be *homogeneous* while the presence of the source term, $S(x,t)$, in (3.7) gives it the label *inhomogeneous*. Hyperbolic pde's can usually be recognized right away because they involve the difference of spatial and temporal partial derivatives being equated to a source function. Wave equations can be second order in their derivatives as these are or any other order as long as the highest order of the time and space derivatives are the same. For example, a first order wave equation known as the *advection* equation is

$$\frac{\partial\phi(x,t)}{\partial x} - a\frac{\partial\phi(x,t)}{\partial t} = 0 \tag{3.8}$$

where a is a constant.

Exercise 3.2.1. *Show that the left side of equation* (3.6) *can be factored into two operators of the form of the left side of equation* (3.8)*. What values does the constant a take in each expression? Show that* $\psi(x,t) = \psi_1(x+vt) + \psi_2(x-vt)$ *is a solution to* (3.6) *by showing that the two factors annihilate* ψ_1 *or* ψ_2*. (*$\psi_1(x+vt)$ *means an arbitrary one-dimensional function of* $x+vt$ *and similarly for* $\psi_2(x-vt)$*). Can you explain the physical meaning of this result?*

3.2.2 An inhomogeneous fluid

labelssec:ihf An inhomogeneous fluid is one whose physical properties vary with position[5]. Let $\rho(\vec{x})$ and $K(\vec{x})$ be the density and bulk modulus of such a fluid. Let $P(\vec{x})$ represent a pressure fluctuation from the ambient pressure P_0 in the fluid. Thus the total fluid pressure is $P(\vec{x}) + P_0$. The bulk modulus is a material property defined by the relation

$$P(\vec{x},t) = -K(\vec{x})\theta(\vec{x},t). \tag{3.9}$$

The quantity θ is the *volume strain* or *dilatation* and is a measure of local fractional volume change. This relation says that the pressure fluctuations are directly proportional to induced volume changes. θ is related to particle velocity, $\vec{\nu}$, though $\partial_t\theta = \vec{\nabla}\cdot\vec{\nu}$. The minus sign in equation (3.9) is needed because an increase in pressure causes a decrease in volume and vice-versa. This is an example of a *constitutive relation* that defines how stress relates to strain. In other words, equation (3.9) is *Hooke's law* for a fluid.

We assume that the fluid is at rest except for those motions induced by the pressure disturbance $P(\vec{x},t)$. As with the vibrating string, the motion of the fluid is defined by Newton's second law

$$-\vec{\nabla}P(\vec{x},t) = \rho(\vec{x})\frac{d\vec{\nu}(\vec{x},t)}{dt}. \tag{3.10}$$

This says that spatially changing pressure causes local forces in the fluid that give rise to local

[4] In this context this is just a fancy way of saying force per unit length.

[5] It is unfortunate that *inhomogeneous* is used in at least two very different contexts in this theory. Here the word refers to physical parameters that vary with position. In the previous section the word referred to a pde with a source term and that usage has nothing to do with this.

particle accelerations. The minus sign is also necessary here because the local forces caused by the pressure gradient point in the direction opposite to the gradient. That is, the gradient points from low to high pressure but the fluid will move from high to low pressure.

The interpretation of the $\frac{d\vec{\nu}}{dt}$ requires some care. Generally in fluid dynamics, such time derivatives consist of two terms as $\frac{dT}{dt} = \frac{\partial T}{\partial t} + (\vec{\nu} \cdot \vec{\nabla})T$ where T denotes any scalar fluid property such as temperature. The first term describes local temporal changes (e.g. local heating) while the second describes changes due to new material being transported to the analysis location by fluid motion. For this reason, the second term is known as the *convective term*. The assumption that the fluid is at rest except for the motions caused by the passing pressure wave allows the convective term to be neglected. If the term is included, a nonlinear wave equation will result. Thus, equation (3.10) is rewritten approximately as

$$-\vec{\nabla} P(\vec{x}, t) = \rho(\vec{x}) \frac{\partial \vec{\nu}(\vec{x}, t)}{\partial t}. \tag{3.11}$$

A wave equation for pressure can be derived by taking the divergence of (3.11),

$$-\vec{\nabla} \cdot \vec{\nabla} P(\vec{x}, t) = \vec{\nabla} \rho(\vec{x}) \cdot \frac{\partial \vec{\nu}(\vec{x}, t)}{\partial t} + \rho(\vec{x}) \frac{\partial \vec{\nabla} \cdot \vec{\nu}(\vec{x}, t)}{\partial t}, \tag{3.12}$$

and substituting equation (3.11) into the first term on the right side and ∂_t of equation (3.9) into the second. This gives

$$\nabla^2 P(\vec{x}, t) - \vec{\nabla} \ln \rho(\vec{x}) \cdot \vec{\nabla} P(\vec{x}, t) - \frac{\rho(\vec{x})}{K(\vec{x})} \frac{\partial^2 P(\vec{x}, t)}{\partial t^2} = 0 \tag{3.13}$$

where $\nabla^2 = \vec{\nabla} \cdot \vec{\nabla}$ and $\frac{1}{\rho} \vec{\nabla} \rho = \vec{\nabla} \ln \rho$ have been used. If the second term were absent in equation (3.13) then we would have a classic scalar wave equation for pressure. This occurs exactly if $\rho(\vec{x})$ is constant. In many other cases when $\rho(\vec{x})$ varies slowly the term will be negligible. Neglecting the $\vec{\nabla} \ln \rho$ term then gives

$$\nabla^2 P(\vec{x}, t) - \frac{1}{v^2} \frac{\partial^2 P(\vec{x}, t)}{\partial t^2} = 0 \tag{3.14}$$

where $v = \sqrt{\frac{K}{\rho}}$. Equation (3.14) is a scalar wave equation that approximately models the propagation of pressure waves through an inhomogeneous fluid. Though a number of assumptions have been made in deriving this result, it still has quite general application. Also, it is possible to use a solution of (3.14) to develop an approximate solution to the more complex Equation (3.13). Let $\hat{P}(\vec{x}, t)$ be a solution to equation (3.14), then it can be used to approximately express the $\vec{\nabla} \ln \rho \cdot \vec{\nabla} P$ term in (3.13) to give

$$\nabla^2 P(\vec{x}, t) - \frac{1}{v^2(\vec{x})} \frac{\partial^2 P(\vec{x}, t)}{\partial t^2} = \vec{\nabla} \ln \rho(\vec{x}) \cdot \vec{\nabla} \hat{P}(\vec{x}, t). \tag{3.15}$$

The term on the right does not depend upon the unknown $P(\vec{x}, t)$ but rather plays the role of a source term in scalar wave equation. Thus the effect of strong density inhomogeneity introduces an approximate source term whose strength is proportional to the logarithmic gradient of density. This process can be repeated indefinitely with the solution from iteration n-1 being used to compute the effective source term for iteration n.

Exercise 3.2.2. *Show that the volume dilatation, θ also satisfies a scalar wave equation.*

Exercise 3.2.3. *Consider the 1-D inhomogeneous fluid. Write down the wave equation that approximately models pressure waves in a long narrow cylinder.*

3.3 Finite difference modelling with the acoustic wave equation

This section describes a simple facility for finite difference modelling using the acoustic wave equation. This toolkit was originally developed by Carrie Youzwishen who was employed by this author for that purpose. Subsequently it has been evolved by this author and Dr. Zhengsheng Yao. The package provides a basic facility, using second order finite difference approximations, for modelling with the variable velocity acoustic wave equation in two dimensions. The facility provides tools for model building, source specification, time-stepping a wavefield, seismogram creation, and making wavefield movies. Sources and receivers can be positioned anywhere within the 2D medium (including on the boundaries) and absorbing boundary conditions are implemented.

Consider the variable-velocity scalar wave equation in two dimensions

$$\nabla^2\psi(x,z,t) = \frac{\partial^2\psi(x,z,t)}{\partial x^2} + \frac{\partial^2\psi(x,z,t)}{\partial z^2} = \frac{1}{v^2(x,z)}\frac{\partial^2\psi(x,z,t)}{\partial t^2}. \tag{3.16}$$

A second-order approximation for the time derivative may be implemented as

$$\nabla^2\psi(x,z,t) = \frac{1}{\Delta t^2 v^2(x,z)}\left[\psi(x,z,t+\Delta t) - 2\psi(x,z,t) + \psi(x,z,t-\Delta t)\right] \tag{3.17}$$

where Δt is the time discretization interval. This expression can be solved for the wavefield at $t+\Delta t$ to give

$$\psi(x,z,t+\Delta t) = \left[2 + \Delta t^2 v^2(x,z)\nabla^2\right]\psi(x,z,t) - \psi(x,z,t-\Delta t). \tag{3.18}$$

This is an expression for *time stepping* the wavefield. It shows that estimation of the wavefield at $t+\Delta t$ requires knowledge of the two earlier wavefields at t and $t-\Delta t$. Each of these wavefields is called a *snapshot* and, in a computer simulation, they are all two dimensional matrices.

Equation (3.18) shows that $\psi(x,z,t+\Delta t)$ is estimated from the superposition of three terms: $2\psi(x,z,t)$, $\Delta t^2 v^2(x,z)\nabla^2\psi(x,z,t)$, and $-\psi(x,z,t-\Delta t)$. Of these, the second requires the computation of the Laplacian (∇^2) on the current wavefield which is an expensive operation. MATLAB supplies the function `del2` that computes $.25\nabla^2$ of a matrix using centered second-order finite operators that are modified near the boundary. Experimentation showed that `del2` was not optimal for use with absorbing boundary conditions so two alternate Laplacians, ***del2_5pt*** and ***del2_9pt***, were created. These functions both pad the entire boundary of the input matrix with extra rows and columns of zeros and this works better with absorbing boundaries. ***del2_5pt*** implements ∇^2 using second-order finite-difference operators while ***del2_9pt*** uses fourth-order. Thus ***del_5pt*** computes the approximation

$$\nabla^2\psi(x,z,t) = \frac{\psi(x+\Delta x,z,t) - 2\psi(x,z,t) + \psi(x-\Delta x,z,t)}{\Delta x^2} + \frac{\psi(x,z+\Delta z,t) - 2\psi(x,z,t) + \psi(x,z-\Delta z,t)}{\Delta z^2} \tag{3.19}$$

and *del2_9pt* computes

$$\nabla^2\psi(x,z,t) =$$
$$\frac{1}{12\Delta x^2}\left[-\psi(x+2\Delta x,z,t)+16\psi(x+\Delta x,z,t)-30\psi(x,z,t)+16\psi(x-\Delta x,z,t)-\psi(x-2\Delta x,z,t)\right]+$$
$$\frac{1}{12\Delta z^2}\left[-\psi(x+2\Delta z,z,t)+16\psi(x+\Delta z,z,t)-30\psi(x,z,t)+16\psi(x-\Delta z,z,t)-\psi(x-2\Delta z,z,t)\right] \tag{3.20}$$

The core function in the finite difference toolbox is *afd_snap* (the '*afd*' prefix stands for *acoustic finite difference*). Function *afd_snap* requires two input wavefields representing $\psi(x,z,t)$ and $\psi(x,z,t-\Delta t)$ and computes $\psi(x,z,t+\Delta t)$ according to equation (3.18). This function is intended to be used in a computation loop that time-increments a wavefield any number of steps. Two initial wavefields must be constructed to start the simulation and sources may be prescribed by placing appropriate impulses in these two wavefields. Receivers are simulated by extracting samples at each receiver location from each $\psi(x,z,t)$ as it is computed. These extracted samples may be accumulated in vectors representing the recorded traces.

The use of equation (3.18) in a time-stepping simulation is known to be unstable in certain circumstances (Mitchell and Griffiths, 1980). Instability means that the amplitudes of the wavefield grow without bound as it is stepped through time. The key to this behavior is the amplitude of the $\nabla^2\psi(x,z,t)$ term in equation (3.18). Using the five-point Laplacian of equation (3.19) (with $\Delta z=\Delta x$) in equation (3.18) leads to

$$\psi(x,z,t+\Delta t)=2\psi(x,z,t)-\psi(x,z,t-\Delta t)+\frac{\Delta t^2v^2}{\Delta x^2}\left[\delta_{xx}\psi(x,z,t)+\delta_{zz}\psi(x,z,t)\right] \tag{3.21}$$

where $\delta_{xx}\psi(x,z,t)=\psi(x+\Delta x,z,t)-2\psi(x,z,t)+\psi(x-\Delta x,z,t)$ and similarly for δ_{zz}. In this expression, all of the ψ terms can be considered to have similar magnitude. Thus, the factor $\Delta t^2v^2\Delta x^{-2}$ is a possible *amplification factor* if it becomes too large. Lines et al. (1999) show that the condition for stability is

$$\frac{v\Delta t}{\Delta x}\le\frac{2}{\sqrt{a}} \tag{3.22}$$

where the constant a is the sum of the absolute values of the weights for the various wavefield terms in the finite-difference approximation for ∇^2. For the Laplacian of equation (3.19), $a=8$ while for equation (3.20), $a=128/12=32/3$. Also, since v is a function of (x,z) it suffices to use the maximum velocity in the model. Thus the stability conditions are

$$\frac{v_{max}\Delta t}{\Delta x}\le\begin{cases}\frac{1}{\sqrt{2}} & \text{second-order Laplacian,}\\ \sqrt{\frac{3}{8}} & \text{fourth-order Laplacian.}\end{cases} \tag{3.23}$$

Thus, the time and space sample rates cannot be chosen independently. Generally, finite-difference operators need many more samples than the Nyquist criterion of two per wavelength. Technically, this is because the operators cause an artificial dispersion called *grid dispersion*. Grid dispersion preferentially affects the shorter wavelengths so oversampling reduces the dispersion. A good rule of thumb is about five to ten samples per wavelength. Typically, in the creation of a model, a desired temporal frequency range is known. Then, the minimum wavelength is given by $\lambda_{min}=v_{min}/f_{max}$ and the spatial sample rate can be chosen to achieve a desired number of

samples-per-wavelength. Finally the temporal sample rate is chosen to achieve stability.

Usually, the user will not invoke *afd_snap* directly. Instead, *afd_shotrec* is provided to create a source record and *afd_exreflector* will create exploding reflector models. The exploding reflector model and *afd_exreflector* will be described in chapter 5 and only *afd_shotrec* will be discussed here. *afd_shotrec* requires inputs giving: the temporal and spatial sample sizes, the maximum record time, the velocity matrix, the receiver positions, the wavelet, the desired Laplacian (five point or nine point), and the two initial snapshots of the wavefield (`snap1` and `snap2`). The snapshots should be initialized to matrices of zeros the same size as the velocity model. Then the source configuration is described by placing appropriate impulses in these two snapshots. A simple strategy is to leave snap1 as all zeros and simply place impulses in snap2.

Code Snippet 3.3.1. *This code illustrates the use of* `afd_shotrec` *to model a shot record. Lines 1-11 build the velocity model and lines 13-25 create a second-order and a fourth-order seismogram. The results are shown in Figures 3.2, 3.3, and 3.4.*

```
1    %make a velocity model
2    nx=128;dx=10;nz=128; %basic geometry
3    x=(0:nx-1)*dx;z=(0:nz-1)*dx;
4    v1=2000;v2=2800;v3=3200;%velocities
5    vmodel=v3*ones(nx,nz);
6    z1=(nz/8)*dx;z2=(nz/2)*dx;
7    dx2=dx/2;
8    xpoly=[-dx2 max(x)+dx2 max(x)+dx2 -dx2];zpoly=[-dx2 -dx2 z1+dx2 z1+dx2];
9    vmodel=afd_vmodel(dx,vmodel,v1,xpoly,zpoly);
10   zpoly=[z1+dx2 z1+dx2 z2+dx2 z2+dx2];
11   vmodel=afd_vmodel(dx,vmodel,v2,xpoly,zpoly);
12
13   dtstep=.001;%time step
14   dt=.004;tmax=1;%time sample rate and max time
15   xrec=x;%receiver locations
16   zrec=zeros(size(xrec));%receivers at zero depth
17   snap1=zeros(size(vmodel));
18   snap2=snap1;
19   snap2(1,length(x)/2)=1;%place the source
20   %second order laplacian
21   [seismogram2,seis2,t]=afd_shotrec(dx,dtstep,dt,tmax, ...
22                            vmodel,snap1,snap2,xrec,zrec,[5 10 30 40],0,1);
23   %fourth order laplacian
24   [seismogram4,seis4,t]=afd_shotrec(dx,dtstep,dt,tmax, ...
25                            vmodel,snap1,snap2,xrec,zrec,[5 10 30 40],0,2);
```

End Code

Code Snippet 3.3.1 illustrates the use of these finite difference facilities to model a single shot record. First, a three-layer velocity model is defined on lines 1 through 11 by calling *afd_vmodel* twice. The velocity model is a matrix representing v_{ins} in (x, z) space. The function *afd_vmodel* is a convenience function that fills in a polygonal area in a matrix with a constant value. The polygonal area is defined by the (x, z) coordinates of its nodes and the coordinate origin is assumed to be the upper left corner of the model. In all of these finite difference codes, the vertical and horizontal sample sizes must be identical. In this example, the velocity model has three layers with horizontal interfaces and layer velocities of 2000 m/s, 2800 m/s, and 3200 m/s.

The model is built by first defining (x, z) coordinates (lines 2-3) and filling the velocity matrix with 3200 m/s (line 5). Subsequently, two polygons are defined to represent the two upper layers. On line 6, `z1` and `z2` are the depths to the bottom of the first and second layers. Line 8 defines the (x, z) coordinates of the four vertices of a rectangle representing the first layer. A property of the

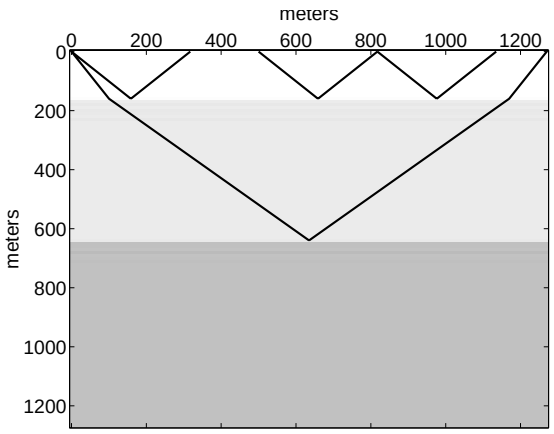

Figure 3.2: The velocity model created in Code Snippet 3.3.1.

algorithm used by *afd_vmodel* is that points that lie exactly on the boundary of the polygon are considered *outside* the polygon and so do not acquire the new velocity. The variable `dx2`, defined on line 7 as half of the grid spacing, is used on line 8 to define a rectangle that is half a grid spacing above depths 0 and `z2` and half a grid spacing outside the minimum and maximum x coordinates. This ensures that the rectangle extends to the limits of the velocity matrix. Line 9 fills this rectangle with the velocity of 2000 m/s and then lines 10 and 11 repeat this process for the next layer. The resulting velocity model is shown in Figure 3.2. This plot was made using `plotimage(vmodel-3000,z,x)`. A constant is subtracted from the velocity model so that the resulting matrix has both positive and negative values as expected by *plotimage*. The raypaths shown in this figure correspond to traveltimes shown in Figures 3.3 and 3.4.

Code Snippet 3.3.1 creates two seismograms, the first (line 21) uses a second-order Laplacian (equation 3.19) and the second (line 23) uses a fourth-order Laplacian (equation 3.20). The preparation for the seismograms defines the time step (line 13), temporal sample rate (line 14), maximum time (line 14), the receiver locations (lines 15-16) and the source strength and geometry (line 19). The time step is generally chosen to be small enough to satisfy the stability requirement (equation 3.23) and the temporal sample rate is usually much more coarse. *afd_shotrec* internally calculates the seismogram at the sample rate of the time step and then resamples it to the desired temporal sample rate. This is sensible because it is a well known property of finite-difference methods that the higher frequencies are physically inaccurate. In this case, the Nyquist frequency for a sample rate of .001 seconds is 1000 Hz while for .004 seconds it is 125 Hz. Considering that the antialias filter for resampling to .004 seconds will attenuate frequencies above about half of Nyquist, this example anticipates using only frequencies below about 60 Hz and that is less than 10% of the original Nyquist.

The variables `snap1` and `snap2` created on lines 17-19 represent the wavefield $\psi(x, z, t = -\Delta t)$ and $\psi(x, z, t = 0)$ respectively. They are created to be the same size as the velocity model and `snap1` is a matrix of zeros. The source is specified by placing appropriate nonzero samples in `snap2`. In this case, a point source in the center of the x axis at $z = 0$ is simulated.

At this stage, all of the input parameters to *afd_shotrec* have been described except for the final three. These specify the filter (or wavelet) to be convolved with the seismogram, the phase of the filter, and the order of the Laplacian. In this case, *Ormsby* filter specifications are given as [5 10 30 40] and this means that the filter pass band begins at 5 Hz., reaches full amplitude at 10 Hz.,

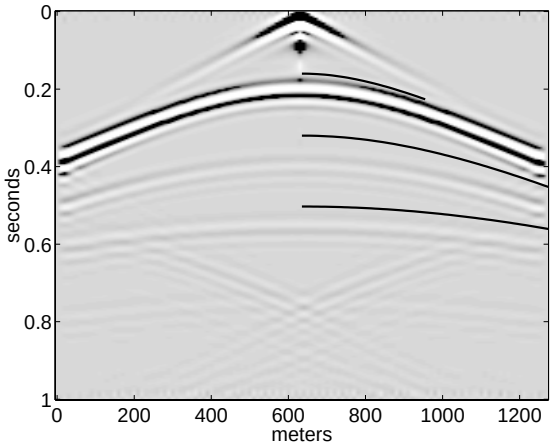

Figure 3.3: The second-order seismogram created on line 21 of Code Snippet 3.3.1. Superimposed on the right-hand side are raytraced traveltimes for (from top to bottom) the first primary reflection, the first-order multiple in the top layer, and the second primary reflection. The corresponding raypaths are shown in Figure 3.2.

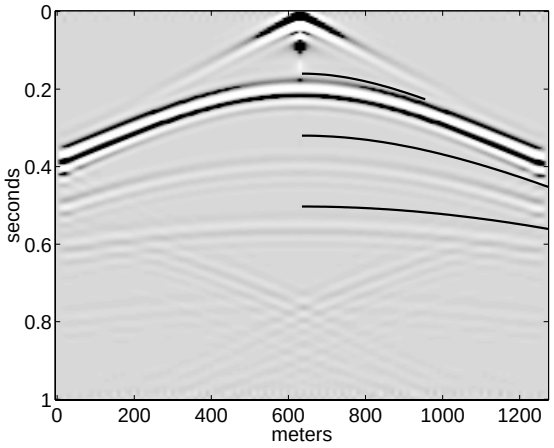

Figure 3.4: The fourth-order seismogram created on line 23 of Code Snippet 3.3.1. See Figure 3.3 for a description of the superimposed traveltimes.

begins to ramp down at 30 Hz., and rejects frequencies above 40 Hz. The penultimate parameter specifies the phase of the filter that is, in this case, zero (a value of 1 gives *minimum phase*). The last parameter is either 1, indicating a second-order Laplacian, or 2, meaning a fourth-order approximation.

The seismograms are shown in Figures 3.3 and 3.4 respectively. Both of these figures are plotted with a highly clipped display, otherwise, the direct wave near the source would be the only visible arrival. Though similar at first glance, these two seismograms are significantly different. The three hyperbolic reflections on each figure are (from the top) the primary reflection off the bottom of the first layer, the first-order multiple reflecting between the bottom of the first layer and the surface, and the primary reflection off the bottom of the second layer. The traveltimes for these events are superimposed on the right-hand sides of the figures and their raypaths are shown in Figure 3.2. The reflections for these events lag significantly behind the raytraced traveltimes but more so on the second-order solution. This time lag occurs because the finite difference approximations to the derivative in the wave equation have a frequency-dependent performance. Essentially, for wavelengths that are large compared to the computation grid, the approximate derivatives are acceptable but, as the wavelength approaches the grid spacing, the approximation becomes very poor. The result is a phenomenon known as grid dispersion, meaning that the actual propagation speed of waves on the grid is not simply $v_{ins}(x, z)$ but is a complex function of wavelength (or, equivalently, frequency). Grid dispersion makes the reflections appear to lag behind the corresponding raytrace traveltimes. It also makes the apparent wavelet appear more elongated (dispersed). Comparison of the first reflection at far offsets on both seismograms shows that the second-order result has a more dispersed waveform. Also, the fourth-order result has a smaller lag behind the raytrace traveltimes than the second-order for all reflections.

Careful inspection of both Figures 3.3 and 3.4 shows a set of events that originate where each reflection meets the sides of the seismogram and then cross in the center of the figures. These are typical artifacts known as *edge effects*. They arise because a computer simulation of wave propagation must always operate in a finite domain with definite boundaries. When a wave encounters such a boundary, it behaves as though it has met a perfect reflector. These boundary reflections would be much worse if `afd_shotrec` did not incorporate *absorbing boundaries* (Clayton and Engquist, 1977). Obviously, absorbing boundaries are not perfect but they do help. The boundary related artifacts can be seen to be generally quite steeply dipping in (x, t). This is because the absorbing boundary conditions are optimized for a small range of wavefront incidence angles around normal incidence. That is, a wavefront that encounters the boundary at normal incidence is completely absorbed while one making an angle of, say, 30° is partially reflected.

3.4 The one-dimensional synthetic seismogram

For a one-dimensional acoustic medium, there exists an algorithm that can generate a complete synthetic seismogram that includes all multiples and the effects of transmission losses. The theory is also very flexible in that it allows the explicit separation of the various physical effects. These 1-D synthetic seismograms are a very important tool in seismic interpretation. After migration, or for nearly horizontal reflectors in any case, the 1-D model is quite appropriate. Comparison of 1-D synthetic seismograms with seismic data, a process called *tieing*, allows reflections from key geologic markers to be identified. Typically, velocity and density information are provided through *well logging* as finely-sampled functions of depth. A common sample rate is 1 foot or .31 meters. The well logging is usually confined to a certain depth zone beginning at the exploration target and extending upwards perhaps hundreds of meters. The near surface is rarely logged and is usually represented as a homogeneous layer whose velocity is selected to optimize the tie between the synthetic seismogram and the seismic data. The theory presented here constructs the response of the 1-D medium to a unit impulse, the *impulse response*, and this must be convolved with an appropriate wavelet before comparison with seismic data. The estimation of the appropriate wavelet is a potentially complex task that is called *wavelet analysis*.

3.4.1 Normal incidence reflection coefficients

Consider a 1-D displacement wave incident from above upon an interface where α_1, ρ_1 and α_2, ρ_2 are the velocities and densities of the upper and lower media. Let the incident wave be described by $f(t - z/\alpha_1)$, the reflected wave by $g(t + z/\alpha_2)$, and the transmitted wave by $h(t - z/\alpha_2)$. Here $f, g,$ and$h$ represent arbitrary functions describing traveling waves. The reflected and transmitted waves can be determined in terms of the incident waves by requiring that the total wavefield satisfy *continuity of displacement* and *continuity of pressure*. The first condition is required so that the interface remains in welded contact during the passage of the wave. The second condition is needed to ensure that local acceleration remains finite at the interface. This is because the local force is determined by the pressure gradient and a discontinuous pressure means an infinite force and that means infinite acceleration.

For definiteness, let the interface be at $z = 0$ and continuity of displacement is expressed by

$$f|_{z=0} + g|_{z=0} = h|_{z=0}. \tag{3.24}$$

To develop a form for continuity of pressure, Hooke's law can be used. As show in equation (3.9), for an inhomogeneous fluid, Hooke's law relates the applied pressure to the negative of the differential

volume change. In 1-D this reduces to $p = -K\partial_z u$ where p is pressure, u is displacement, and K is the bulk modulus. In the development of equation (3.14) it was shown that the wave velocity is given by the square root of the bulk modulus divided by the density, or equivalently, $K = \rho\alpha^2$. Thus continuity of pressure can be expressed by

$$\rho_1\alpha_1^2 \left.\frac{\partial f}{\partial z}\right|_{z=0} + \rho_1\alpha_1^2 \left.\frac{\partial g}{\partial z}\right|_{z=0} = \rho_2\alpha_2^2 \left.\frac{\partial h}{\partial z}\right|_{z=0}. \tag{3.25}$$

Since $f = f(t - zalpha_1)$, then $\partial_z f = -\alpha_1^{-1} f'$ where f' is the derivative of f with respect to its entire argument. With similar considerations for g and h, equation (3.25) becomes

$$\rho_1\alpha_1 f'|_{z=0} + \rho_1\alpha_1 g'|_{z=0} = \rho_2\alpha_2 h'|_{z=0} \tag{3.26}$$

which can be immediately integrated to give

$$\rho_1\alpha_1 f|_{z=0} + \rho_1\alpha_1 g|_{z=0} = \rho_2\alpha_2 h|_{z=0}. \tag{3.27}$$

Considering the incident wavefield as known, equations (3.24) and (3.27) constitute two linear equations for the two unknowns g and h. Defining the *acoustic impedance* $I = \rho\alpha$ the solution can we written as

$$\begin{aligned} g|_{z=0} &= -Rf|_{z=0} \\ h|_{z=0} &= Tf|_{z=0} \end{aligned} \tag{3.28}$$

where the *reflection coefficient*, R, and the *transmission coefficient*, T are given by

$$R = \frac{I_2 - I_1}{I_2 + I_1} \tag{3.29}$$

$$I = \frac{2I_1}{I_2 + I_1}. \tag{3.30}$$

The reflection coefficient is defined such that a transition from lower to higher impedance is a positive R. It is easily shown that $R + T = 1$. A wave incident from below on the same interface will experience a reflection coefficient of $-R$ and a transmission coefficient of $I_2T/I_1 = 1 + R$. As the first of equations (3.28) shows, the reflected wave is actually $-R$ times the incident wave. This is because these are displacement waves and the direction of particle displacement reverses at a positive impedance contrast. This can be understood by considering the limiting case of $I_2 \to \infty$ corresponding to incidence upon a perfectly rigid material. In this case, $T \to 0$ and continuity of displacement requires that the reflected wave have equal and opposite displacement to that of the incident wave.

Exercise 3.4.1. ?? *By directly computing pressures, show that the reflected pressure is $+R$ times the incident pressure. Also show that the transmitted pressure is $+T$ times the incident pressure. Thus the displacement and pressure waves have the same transmission equation but there is a sign difference in their reflection expressions. Explain these results with a physical description. (Hint: If the particle displacement is in same direction as the wave propagation, then is the pressure a compression or a rarefaction? What about if the displacement is in the opposite direction to the direction of wave propagation.? Is a compression a positive or negative pressure? Why?)*

An alternative form for the reflection and transmission coefficients involves writing them in terms of the contrast and average of impedance across the layer. Define $I = (I_1 + I_2)/2$ and $\Delta I = I_2 - I_1$

and equations (3.28) become

$$R = \frac{\Delta I}{2I}$$
$$T = \frac{I - .5\Delta I}{I}. \tag{3.31}$$

Using $I = \rho\alpha$ allows R to be re-expressed as

$$R = \frac{1}{2}\left[\frac{\Delta\alpha}{\alpha} + \frac{\Delta\rho}{\rho}\right] \tag{3.32}$$

which expresses the contributions of both α and ρ to the reflection coefficient.

3.4.2 A "primaries-only" impulse response

Consider a layered 1-D medium described by wave velocity $\alpha(z)$ and density $\rho(z)$ for $z \geq 0$. In order to apply the results of the previous section, let $\alpha(z)$ and $\rho(z)$ be approximated by a discrete sequences $\{\alpha_k\}$ and $\{\rho_k\}$, $k = 1, 2, \ldots N$ representing a stack of homogeneous layers having thicknesses of $\{\Delta z_k\}$. Furthermore, let the thicknesses be chosen such that

$$\frac{\Delta z_k}{\alpha_k} = \frac{\Delta t}{2} = \text{constant} \tag{3.33}$$

where Δt is a constant giving the two-way traveltime across any layer. This requirement is not strictly necessary for the theory but is convenient for algorithm development. The condition means that homogeneous intervals of high velocity will be represented by more samples than corresponding intervals of low velocity. It is quite acceptable to have many layers with nearly zero impedance contrasts. By choosing Δt sufficiently small, $\alpha(z)$ and $\rho(z)$ can be approximated with arbitrary accuracy.

The constant Δt becomes the temporal sample rate of the output seismogram. However, greater realism is obtained by choosing Δt much smaller than the desired sample rate and resampling the seismogram after it is computed.

Given this discrete approximation to the desired model, it is possible to explicitly calculate all primary reflections. For each layer, let the sequence $\{R_k\}$ denote the reflection coefficient at the layer bottom for incidence from above. Let a co-incident source-receiver pair be placed at $z = 0$ and let the source emit a unit impulse *of displacement* at $t = 0$. The reflection from the first interface will have magnitude $-R_1$ and will arrive at the receiver at time Δt. The minus sign arises from equation (3.28) an occurs because the reflection reverses the direction of displacement. A z transform expression conveniently expresses the first arrival as $-R_1 z^{\Delta t}$. The transmitted wave, of amplitude $1 - R_1$ crosses the first interface and reflects with reflection coefficient $-R_2$. It re-crosses the first interface from below with a transmission coefficient of $1 + R_1$ and then arrives at the receiver. This second reflection has a z transform of $-(1 - R_1)R_2(1 + R_2)z^{2\Delta t} = -(1 - R_1^2)R_2 z^{2\Delta t}$ so the net transmission effect of two-way passage through the first interface is $1 - R_1^2$. Similar reasoning leads to the z transform of the third reflection as $-(1 - R_1^2)(1 - R_2^2)R_3 z^{3\Delta t}$ and for the general j^{th} arrival $-\prod_{k=1}^{j-1}(1 - R_k^2)]R_j z^{j\Delta t}$. A primaries-only synthetic seismogram of displacement for N reflectors is

represented by the sum of these z transform terms as

$$u_{imp}(z) = 1 - R_1 z^{\Delta t} - (1-R_1^2)R_2 z^{2\Delta t} - (1-R_1^2)(1-R_2^2)R_3 z^{3\Delta t} - \ldots \\ - \left[\prod_{k=1}^{j-1}(1-R_k^2)\right] R_j z^{j\Delta t} - \cdots - \left[\prod_{k=1}^{N-1}(1-R_k^2)\right] R_n z^{n\Delta t} \quad (3.34)$$

where the subscript *imp* indicates "impulse response" and the leading 1 represents a direct arrival. This expression has the compact representation

$$u_{imp}(z) = 1 - \sum_{j=1}^{N}\left[\prod_{k=1}^{j-1}(1-R_k^2)\right] R_j z^{j\Delta t}. \quad (3.35)$$

The term in square brackets is called the *transmission loss*. Often seismic data has had a correction applied for transmission loss, in which case it is useful to construct a synthetic seismogram without transmission losses by setting these terms to unity. The result is the simplest model of a seismogram as reflection coefficients arriving at the two-way traveltime to each reflector.

A source waveform other than a unit impulse is easily accounted for. If $W(z)$ represents the z transform of the source waveform, then, for example, the second reflection is given by $(1 - R_1^2)R_2 W(z)z^{2\Delta t}$. That is, $W(z)$ simply multiplies the z transform of the impulsive arrival. For example consider a causal waveform with $W(z) = w_0 + w_1 Z^{\Delta t} + w_2 z^{2\Delta t} + \ldots$. Then the reflection and transmission effects scale each sample of $W(z)$ equally and the later samples simply arrive superimposed on top of later reflections. Of course, this is just a physical way of visualizing a convolution. Thus it is concluded that the source waveform is included in the synthetic seismogram by convolving it with impulse response (e.g. equation (3.35)) of the layered medium. Written with z transforms, this is simply

$$S(z) = u_{imp}(z)W(z). \quad (3.36)$$

It may also be of interest to compute a *pressure* seismogram as this models what is measured by a hydrophone. As discussed in exercise **??**, the essential difference is that the pressure does not change sign upon reflection as pressure does. The transmission effect is the same for both pressure and displacement. Thus, for a the pressure response to a unit impulse of pressure, equation (3.28) should be modified to read

$$p_{imp}(z) = 1 + \sum_{j=1}^{N}\left[\prod_{k=1}^{j-1}(1-R_k^2)\right] R_j z^{j\Delta t}. \quad (3.37)$$

3.4.3 Inclusion of multiples

Waters (1981) gives a computational scheme that allows the calculation of all primaries and multiples up to any desired order. This is essentially a bookkeeping method that uses the diagram of Figure 3.5 to keep track of all waves. The vertical axis of this diagram is model traveltime, τ, while the horizontal axis is event arrival time, t. Layer boundaries are indicated by horizontal lines and indexed by the integer $n = 0, 1, 2, \ldots N$ with 0 denoting the recording surface. As in the previous section, it is assumed that the model has been sampled such that the layers all have the same interval traveltime, $\tau_n - \tau_{n-1}$, and hence are of variable thickness in z. The diagonal lines slanting down to the right correspond to downward traveling waves that are indexed by $j = 1, 2, 3, \ldots$. Similarly, the diagonals that slant up to the right denote upward traveling waves using the index $k = 1, 2, 3, \ldots$.

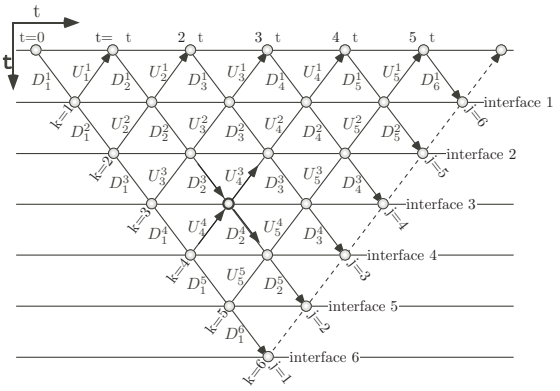

Figure 3.5: The computation triangle for a 6 layer synthetic seismogram. Computation begins with a unit impulse in the upper left corner. Horizontal lines are interfaces between layers, diagonals slanting down from left-to-right are downgoing waves, and diagonals slanting up from left-to-right are upgoing waves. Nodes are denoted by their interface number, n, and their upgoing wave index k. The highlighted node is $(n,k) = (3,4)$.

Though these waves are symbolized by diagonal lines, this is a 1D model so all wave motion is actually vertical. The intersection of a downward wave and an upward wave occurs at a node that corresponds to a *scattering event*. A node will be designated by the pair (n,k) that denote its interface number and the upward traveling wave index. The downward wave index at node (n,k) is $j = k - n + 1$.

The algorithm description requires that the upward and downward traveling waves be described by both their wave indices and the layer number. The layer number is the same as the interface number for the layer bottom. Thus the upward traveling wave in layer n with wave index k will be denoted U_k^n while the j$^{\text{th}}$ downward wave in the same layer will be D_j^n. The fourth upward wave in the first layer is U_4^1 and so on. Considering a particular node (n,k), there are two incident waves, U_k^{n+1} and D_{k-n+1}^n, and two scattered waves, U_k^n and D_{k-n+1}^{n+1}. For example, the second node on the third layer in Figure 3.5 is node $(3,4)$. The incident waves enter the node from the left and are U_4^4 and D_2^3 while the scattered waves exit the node from the right and are U_4^3 and D_2^4. For interface n, denoting the reflection coefficient for incidence from above by R_n, the scattered waves *of displacement* at node (n,k) are related to the incident waves by the equations

$$\begin{aligned} U_k^n &= [1+R_n]\, U_k^{n+1} - R_n D_{k-n+1}^n \\ D_{k-n+1}^{n+1} &= [1-R_n]\, D_{k-n+1}^n + R_n U_k^{n+1}. \end{aligned} \tag{3.38}$$

For pressure waves, the corresponding equations are

$$\begin{aligned} Up_k^n &= [1+R_n]\, Up_k^{n+1} + R_n Dp_{k-n+1}^n \\ Dp_{k-n+1}^{n+1} &= [1-R_n]\, Dp_{k-n+1}^n - R_n Up_k^{n+1} \end{aligned} \tag{3.39}$$

where Up_k^n is the k$^{\text{th}}$ upward traveling pressure wave in the n$^{\text{th}}$ layer and Dp_k^n is a similar downward traveling pressure.

The displacement seismogram algorithm begins by assuming $D_1^1 = 1$ as is appropriate for an impulse response. At node $(1,1)$ the upward incident wave, U_1^2 is zero and so the scattered waves

are $D_1^2 = [1 - R_1]\, D_1^1 = 1 - R_1$ and $U_1^1 = -R_1$. In general, $U_k^n = 0$ if $n < k$ or $n < 1$ and $D_j^n = 0$ if $n < 1$ or $j < 1$. Then U_1^1 can be propagated up to the surface where the scattered wave $D_2^1 = R_0 U_1^1$, *a surface-related multiple*, is calculated. Since all of the layers have the same 2-way traveltime, Δt, the waves arrive at interface 0 at regular intervals where they are accumulated into the seismogram. If surface-related multiples are to be included, the k$^{\text{th}}$ sample of the seismogram is $U_k^1 + D_{k+1}^1 = [1 + R_0]\, U_k^1$, otherwise it is just U_k^1.

Having propagated U_1^1 to the surface and calculated D_1^2 and D_2^1, the upward wave U_2^2 can be calculated and propagated to the surface. U_2^2 is easy and is just $U_2^2 = -R_2 D_1^2 = -R_2\,[1 - R_1]$. Then at node $(1, 2)$, D_2^1 and U_2^2 are incident and the scattered waves are $U_2^1 = [1 + R_1]\, U_2^2 - R_1 D_2^1$ and $D_2^2 = [1 - R_1]\, D_2^1 + R_1 U_2^2$. The last step in bringing U_2 to the surface is to calculate $D_3^1 = R_0 U_2^1$. Thus the second sample on the seismogram is determined and the downgoing waves D_1^3, D_2^2, and D_3^1 are all computed in preparation for propagation of U_3^3 to the surface.

The pattern should now be clear. The wave $U_k^k = -R_k D_1^k$ is propagated to the surface using equations (3.38) to include the contributions from all of the downgoing multiples and to calculate the upgoing and downgoing scattered waves. The propagation of U_k^k to the surface results in the calculation of all of the downgoing waves that are required for U_{k+1}^{k+1}. The process then continues for as long as is desired. In principle, there are infinitely many arrivals from an N layered system because there can be infinitely many multiple bounces.

Using z transform notation, the first few samples of the displacement seismogram can be shown to be

$$u_{imp}(z) = 1 - [1 + R_0]\, R_1 z^{\Delta t} - [1 + R_0]\left[R_2(1 - R_1^2) - R_1^2 R_0\right] z^{2\Delta t} - [1 + R_0]\left[R_3(1 - R_2^2)(1 - R_1^2) - R_2 R_1 (R_2 + R_0)(1 - R_1^2) - R_2 R_1 R_0 (1 - R_1^2) + R_1^3 R_0^2\right] z^{3\Delta t} + \ldots. \quad (3.40)$$

The equivalent pressure seismogram is given by

$$p_{imp}(z) = 1 + [1 - R_0]\, R_1 z^{\Delta t} + [1 - R_0]\left[R_2(1 - R_1^2) - R_1^2 R_0\right] z^{2\Delta t} + [1 - R_0]\left[R_3(1 - R_2^2)(1 - R_1^2) - R_2 R_1 (R_2 + R_0)(1 - R_1^2) - R_2 R_1 R_0 (1 - R_1^2) + R_1^3 R_0^2\right] z^{3\Delta t} + \ldots. \quad (3.41)$$

The displacement and pressure seismograms have difference physical units that are not reflected in equations (3.40) and (3.41).

The increasing complexity of each term is apparent even with only four samples written explicitly. In each term, the primary arrival can be identified by comparison with equation (3.35). In general, there are many possible multiples that also contribute to each term. In equation (3.40), each term after the leading one contains the common factor $1 + R_0^2$ that includes the surface-related multiple. In equation (3.41) the surface-related multiple is included via the term $1 - R_0^2$. If the surface is modelled as a *free surface* then it has $R_0 = 1$ (recall that these reflection coefficients are defined for incidence from above) so each term in the displacement seismogram is doubled while the pressure seismogram is identically zero. This is correct and consistent with the boundary condition of a liquid free surface that calls for the pressure to vanish.

Though the pressure seismogram of equation (3.41) vanishes identically on a free surface, the expression can be used to draw a valuable observation in comparison with the displacement seismogram of equation (3.40). If the two seismograms are added, then the signs are such that the primaries cancel and only the surface-related multiples remain. Or, more importantly, the seismograms can be subtracted to cancel all of the surface-related multiples. Of course, this observation overlooks the fact that the pressure and displacement seismograms have different physical dimensions. In practice, this could be accounted for by scaling the first arrivals to have the same magnitude. These details

aside, the important conclusion is that downgoing waves, like the surface related multiple, can be eliminated by subtracting a scaled pressure seismogram from a displacement seismogram. This observation is the basis of the modern technique of marine seismic surveying with a cable laid on the ocean bottom. Called *OBC recording*, this method includes a paired hydrophone and geophone at each recording position.

More specifically, it is of interest to explicitly detail a few downgoing and upgoing waves for both displacement and pressure. For example, the downgoing waves needed to propagate U_3^3 to the surface are

$$\begin{aligned} D_1^3 &= (1-R_2)(1-R_1) \\ D_2^2 &= -R_1(1-R_1)(R_0+R_2) \\ D_1^3 &= -R_2R_1(1-R_1^2)+R_1^2R_0^2 \end{aligned} \tag{3.42}$$

and the corresponding pressure waves identical in form

$$\begin{aligned} Dp_1^3 &= (1-R_2)(1-R_1) \\ Dp_2^2 &= -R_1(1-R_1)(R_0+R_2) \\ Dp_1^3 &= -R_2R_1(1-R_1^2)+R_1^2R_0^2. \end{aligned} \tag{3.43}$$

Then, propagation of the third upgoing wave to the surface gives, for displacement,

$$\begin{aligned} U_3^3 &= -R_3(1-R_2)(1-R_1) \\ U_3^2 &= -R_3(1-R_2^2)(1-R_1)+R_2R_1(1-R_1)(R_0+R_2) \\ U_3^1 &= -R_3(1-R_2^2)(1-R_1^2)+R_2R_1(1-R_1^2)(R_0+R_2)+R_2R_1R_0(1-R_1^2)-R_1^3R_0^2 \end{aligned} \tag{3.44}$$

and for pressure

$$\begin{aligned} Up_3^3 &= R_3(1-R_2)(1-R_1) \\ Up_3^2 &= R_3(1-R_2^2)(1-R_1)-R_2R_1(1-R_1)(R_0+R_2) \\ Up_3^1 &= R_3(1-R_2^2)(1-R_1^2)-R_2R_1(1-R_1^2)(R_0+R_2)-R_2R_1R_0(1-R_1^2)+R_1^3R_0^2. \end{aligned} \tag{3.45}$$

A term-by-term comparison of these upgoing wave expressions shows that a particular upgoing displacement wave is opposite in sign for every term when compared with the corresponding pressure wave. Though these calculations can be complex, the sign differences are a simple consequence of the scattering equations (3.38) and (3.39) and of the fact that the source was placed above the entire model. Thus, if the initial downgoing displacement and pressure pulses are in phase, then the only upgoing waves are generated by reflections and these have opposite polarity for pressure versus displacement.

OBC recording can be modelled using the one-dimensional seismogram theory by calculating the recorded wave at the particular interface representing the ocean bottom. There, the pressure wave will not trivially vanish and the upgoing waves will have the polarity relationships just described. This can be done with the present algorithm by representing the water layer as many thin layers of equal traveltime and no impedance contrast. However, for a deep water layer and a typical seismogram sample rate of .001 seconds, this can mean a great many layers that serve no purpose other than to give the correct traveltime. In a later section, a synthetic seismogram algorithm will be given that allows the layers to have arbitrary thicknesses. Then it will be possible to model a

water layer with a single algorithmic layer.

As a final comment, this theory makes clear that the effect of multiple reflections is generally *nonstationary*. This means that the multiple content of the seismogram generally increases with time. Ordinary deconvolution theory, which assumes a stationary seismogram, cannot readily deal with a nonstationary multiple sequence. However, there are certain classes of multiples that have the desired stationarity. For example, in equation (3.40) the effect of the free surface is to cause every term after the direct arrival to have the same $1+R_0$ multiplicative factor. This means that the free surface multiples can be modelled as a convolution and that deconvolution might be expected to remove them. A similar effect is true for a bright reflector that occurs deeper in a model. All reflections from beneath the bright reflector will have the same characteristic multiple while those from above it will not.

Exercise 3.4.2. *On a replica of Figure 3.5, sketch the raypath for each multiple that arrives at $t = 3\Delta t$. How many multiples are there in total?*

Exercise 3.4.3. *Verify the expression given for U_3^1 in equation* (3.40)*. Also compute the downgoing waves D_1^4, D_2^3, D_3^2, and D_4^1 in preparation for the next exercise. Verify the downgoing waves to be*

$$D_1^4 = (1-R_3)(1-R_2)(1-R_1)$$
$$D_2^3 = -R_1(1-R_2)(1-R_1)(R_0+R_2) - R_3R_2(1-R_2)(1-R_1)$$
$$D_3^2 = -R_2R_0(1-R_1)(1-R_1^2) + R_1^2R_0^2(1-R_1) - R_3R_1(1-R_2^2)(1-R_1) + R_2R_1^2(1-R_1)(R_0+R_2)$$
$$D_4^1 = -R_3R_0(1-R_2^2)(1-R_1^2) + R_2R_1R_0(1-R_1^2)(R_0+R_2) + R_2R_1R_0^2(1-R_1^2) - R_1^3R_0^3.$$

Exercise 3.4.4. *Calculate U_4^1 using the method described above with Figure 3.5 and the results of the previous exercise. This computation is algebraically tedious. The point of the exercise is to improve understanding of the complexity of the seismogram. The answer is*

$$U_4^1 = -R_4(1-R_3^2)(1-R_2^2)(1-R_1^2) + (1-R_2^2)(1-R_1^2)\left[2R_3R_1R_0 + 2R_3R_2R_1 + R_3^2R_2\right] + (1-R_1^2)\left[R_2^2R_0(1-R_1^2) - 3R_2R_1^2R_0^2 - 2R_2^2R_1^2R_0 - R_2^3R_1^2\right] + R_1^4R_0^3.$$

3.5 MATLAB tools for 1-D synthetic seismograms

There are a number of different possibilities for creating synthetic seismograms in MATLAB. The intended purpose of the synthetic usually dictates which facility to use. For example, the testing of spiking deconvolution algorithms usually is done with a random reflectivity, no multiples, and a minimum phase wavelet while testing predictive deconvolution will usually involve the inclusion of some form of multiples. Alternatively, a synthetic seismogram intended for comparison with processed seismic data will usually be created using a well-log reflectivity, no multiples, and a zero phase wavelet.

3.5.1 Wavelet utilities

Once an impulse-response seismogram has been created, it is usually convolved with a wavelet to simulate the bandlimiting imposed by the seismic source. There are a number of different wavelets available including

`ormsby` Creates an Ormsby bandpass filter.

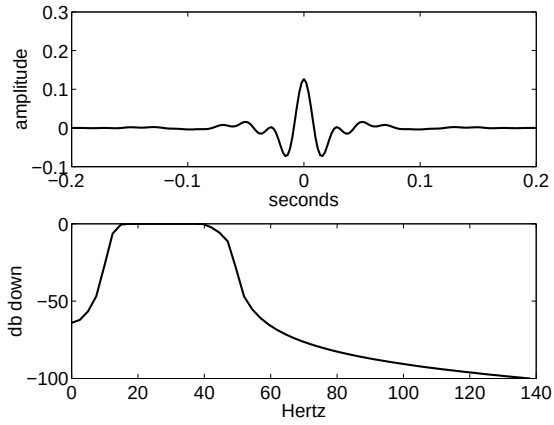

Figure 3.6: An Ormsby wavelet for the passband $10, 15, 40, 50$Hz. is shown in the time domain (top) and its amplitude spectrum is shown against a decibel scale in the bottom frame. This figure was created by Code Snippet 3.5.1

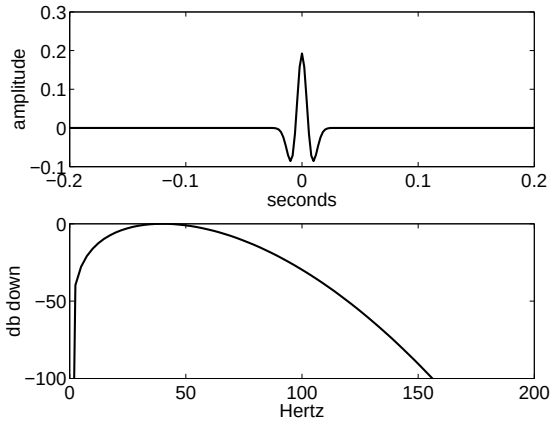

Figure 3.7: A Ricker wavelet with a 40 Hz dominant frequency is shown in the time domain (top) and its amplitude spectrum on a decibel scale is shown in the bottom frame. This was created by Code Snippet 3.5.2

ricker Creates a Ricker wavelet.

sweep Generate a linear Vibroseis sweep.

wavemin Creates a minimum phase wavelet for impulsive sources.

wavevib Creates a vibroseis (Klauder) wavelet.

wavez Creates a zero phase wavelet with a dominant frequency.

These functions all have a similar interface in that they accept a number of input parameters describing the construction of the wavelet and return only two outputs: the wavelet and its time-coordinate vector. Since only ***wavemin*** and ***sweep*** generate causal responses, the time-coordinate vector is essential to properly position the wavelet.

Code Snippet 3.5.1. *This code creates the Ormsby wavelet shown in Figure 3.6. The inputs to* `ormsby` *are the four frequencies defining the passband, the wavelet length, and the sample rate.*

```
1    %make ormsby wavelet
2    [w,tw]=ormsby(10,15,40,50,.4,.002);
3    %compute spectrum
4    [W,f]=fftrl(w,tw);
5    W=todb(W);
```

End Code

The ***ormsby*** command, used in Code Snippet 3.5.1, produces a popular zero-phase wavelet that has an easy specification of its passband. The analytic expression for the Ormsby wavelet is

$$w(t)_{ormsby} = \frac{\pi f_4^2}{f_4 - f_3}\left[\frac{\sin \pi f_4 t}{\pi f_4 t}\right]^2 - \frac{\pi f_3^2}{f_4 - f_3}\left[\frac{\sin \pi f_3 t}{\pi f_3 t}\right]^2 - \frac{\pi f_2^2}{f_2 - f_1}\left[\frac{\sin \pi f_2 t}{\pi f_2 t}\right]^2 + \frac{\pi f_1^2}{f_2 - f_1}\left[\frac{\sin \pi f_1 t}{\pi f_1 t}\right]^2 \tag{3.46}$$

where $f_1 < f_2 < f_3 < f_4$ are specifications of the frequency passband in Hertz. Approximately, the passband begins at f_1 and increases to full amplitude at f_2. It remains at full amplitude until f_3 and ramps down to the stopband at f_4. Figure 3.6 shows an Ormsby wavelet with $(f_1, f_2, f_3, f_4) = (10, 15, 40, 50)$ in the time domain and its Fourier amplitude spectrum. On line 2 in Code Snippet 3.5.1 the wavelet passband is specified as the first four input parameters, the fifth parameter is the desired wavelet length in seconds, and the sixth parameter is the temporal sample rate also in seconds. It is obviously important to ensure that the Nyquist frequency, $f_{Nyquist} = 1/(2\Delta t)$, is greater than f_4. Less obviously, the temporal length of the wavelet must be sufficient to ensure that the character of the wavelet is captured. Since the frequency bandwidth and the temporal width are inversely proportional, a narrow passband wavelet will be temporally long and vice-versa. With any of these wavelet tools, it is always a good idea to plot the wavelet before using it and assess the adequacy of the temporal length. If the length is chosen too short, then truncation effects will distort the desired spectrum. Line 4 of Code Snippet 3.5.1 computes the wavelet's spectrum using *fftrl* and line 5 computes the decibel amplitude spectrum using the utility *todb*. Plotting the real part of the output from *todb* versus `f` creates the spectrum shown in Figure 3.6.

An important consideration when creating wavelets is the question of normalization. That is, how should the overall amplitude of the wavelet be determined. If equation (**??**) is used directly, the maximum amplitude of the wavelet can be quite large. This may be an annoyance in that any time series that is convolved with such a wavelet will have a large change in overall amplitude. An easy solution might seem to be to scale the wavelet such that its maximum absolute value is unity. However, another possibility is to scale the wavelet such that a sinusoid of the dominant frequency is passed at unit amplitude under convolution. These criteria are not identical with the first tending to preserve the peak values of the signal before and after convolution and the second preserves the amplitude of the signal near the wavelet's dominant frequency. The wavelets presented here all use the second method of normalizing with respect to the dominant frequency. However, if other behavior is desired, the wavelet can always be renormalized after it is created. The function *wavenorm* is useful for this purpose.

Code Snippet 3.5.2. *This code creates the Ricker wavelet shown in Figure 3.7. The inputs to* `ricker` *are the temporal sample rate (seconds), the dominant frequency (Hz), and the temporal length (seconds).*

```
1    %make ricker wavelet
2    [w,tw]=ricker(.002,40,.4);
3    %compute spectrum
4    [W,f]=fftrl(w,tw);
5    W=todb(W);
```

End Code

Though the amplitude spectrum of the Ormsby wavelet of Figure 3.6 shows the desired passband, the wavelet is often felt to have an unacceptable level of ripple. The Ricker wavelet has a simpler from in the time-domain though this comes at the expense of a broader, less controlled, passband. The Ricker wavelet is given analytically by

$$w(t)_{ricker} = \left[1 - 2\pi^2 f_{dom}^2 t^2\right] e^{-\pi^2 f_{dom}^2 t^2} \tag{3.47}$$

which can be shown to be the second derivative of a Gaussian. In this expression, f_{dom} is the *dominant frequency* of the wavelet. The Fourier transform of this wavelet is known to be (Sheriff and Geldart, 1995)

$$W(f) = \frac{2f^2}{\sqrt{\pi} f_{dom}^2} e^{-f^2/f_{dom}^2}. \tag{3.48}$$

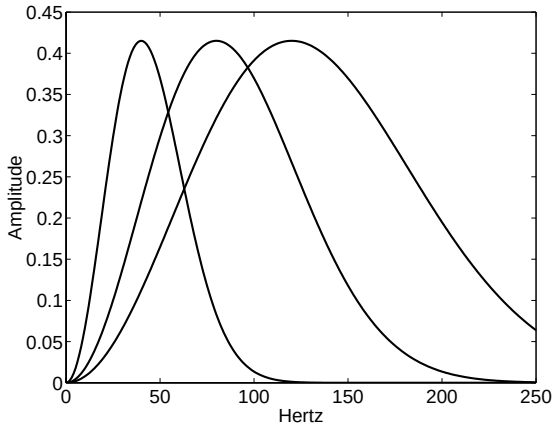

Figure 3.8: Three Ricker wavelet spectra are shown against a linear scale for $f_{dom} = 40, 80,$ and 120 Hz. This figure was created by Code Snippet 3.5.3 using equation (3.48)

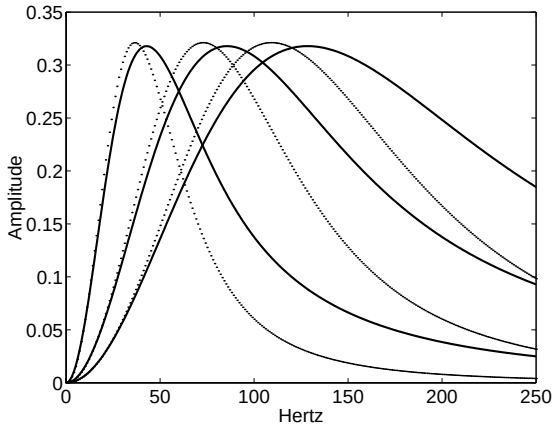

Figure 3.9: Six amplitude spectra are shown illustrating the behavior of equation (3.49). Three dominant frequencies are shown, $f_{dom} = 40, 80, 120$ and the bold curves correspond to $m = 2$ while the dashed curves are for $m = 3$. This was created by Code Snippet 3.5.3

This is a Gaussian multiplied by f^2/f_{dom}^2. It is characteristic of Ricker wavelets that the higher the dominant frequency, the broader the bandwidth. This is evident from equation (3.48) in that the effective width of the Gaussian is seen to be f_{dom}. The effect is demonstrated by Code Snippet 3.5.3 and shown in Figure 3.8.

Code Snippet 3.5.3. *This code calculates the Ricker wavelet spectra shown in Figure 3.8 using equation* (3.48).

```
1    f=0:250;
2    fdom1=40;fdom2=80;fdom3=120;
3    R1=exp(-f.^2/(fdom1^2)).*(2*f.^2)/(sqrt(pi)*fdom1^2);
4    R2=exp(-f.^2/(fdom2^2)).*(2*f.^2)/(sqrt(pi)*fdom2^2);
5    R3=exp(-f.^2/(fdom3^2)).*(2*f.^2)/(sqrt(pi)*fdom3^2);
```

End Code

The Ormsby and Ricker wavelets are zero phase (i.e. symmetric) and are most suitable for models to be compared to fully processed seismic images. An alternative are minimum-phase wavelets that are suitable for models to be compared with raw seismic data or to be used for deconvolution testing. One effective way to design minimum-phase wavelets is to specify the locations of the poles and zeros of the wavelet's z-transform directly. This is the approach taken by Claerbout in **?** and has the advantage that the minimum phase property is easily assured. On the other hand, it is not necessarily easy to control the spectral shape. With this in mind, an alternative is presented here that specifies the wavelet's amplitude spectrum and calculates the minimum-phase wavelet using the *Levinson recursion*. The Levinson recursion is an efficient way to solve a system of linear equations that has *Toeplitz* symmetry. It is discussed more completely in the conjunction with Wiener deconvolution in Chapter **??**. For now, it is just presented as an abstract numerical technique.

Since a minimum-phase wavelet is meant to model an impulsive source, an analytic model for the amplitude spectrum should resemble the observed spectrum for the dynamite blast that is common

in exploration seismology. Such spectra have a dominant frequency below about 30 Hz with larger charges giving lower dominant frequencies and show slow decay towards the high frequencies. The model spectrum adopted here is given by

$$A(f) = \frac{1 - e^{-(f/f_{dom})^2}}{1 + (f/f_{dom})^m} \tag{3.49}$$

where m determines the rate of decay at high frequencies. This spectrum is produced by ***tntamp***. Figure 3.9 was created using ***tntamp*** and shows the behavior of this equation for $m = 2$ and $m = 3$. Clearly, larger m gives a more sharply bandlimited spectrum and this means a longer temporal wavelet. When using $m > 2$, care must be taken that a sufficient temporal length is selected because truncation effects can destroy the minimum-phase property.

Code Snippet 3.5.4. *This code is an excerpt from* `wavemin` *and illustrates the calculation of a minimum-phase wavelet. The column vector of frequencies is* `f`*, the intended length of the wavelet is* `tlength`*, the dominant frequency is* `fdom`*, and the temporal sample rate is* `dt`*.*

```
1    % create the power spectrum
2      powspec=tntamp(fdom,f,m).^2;
3    % create the autocorrelation
4      auto=ifftrl(powspec,f);
5    % run this through Levinson
6      nlags=tlength/dt+1;
7      b=[1.0 zeros(1,nlags-1)]';
8      winv=levrec(auto(1:nlags),b);
9    % invert the wavelet
10     wavelet=real(ifft(1. ./(fft(winv))));
11     twave=(dt*(0:length(wavelet)-1)';
12   % now normalize the wavelet
13     wavelet=wavenorm(wavelet,twave,2);
```

End Code

Function ***wavemin*** invokes ***tntamp*** to create a minimum phase wavelet as is shown in Code Snippet 3.5.4. Line 2 creates the desired power spectrum by squaring the output from ***tntamp***. On line 4, the power spectrum is inverse Fourier transformed to create the autocorrelation of the desired wavelet. Lines 6-8 set up a linear system and solve it using the Levinson recursion. This linear system is derived in Chapter **??** where it is shown to specify the minimum-phase inverse to a minimum-phase wavelet. It is of the form $\underline{\underline{A}}\underline{x} = \underline{b}$ where $\underline{\underline{A}}$ is a Toeplitz matrix formed from the autocorrelation, $\underline{x}$ is the unknown minimum-phase inverse, and $\underline{b}$ is a column vector of zeros except for a one in the first place. Solving this system with ***levrec*** does not require the formation of the matrix $\underline{\underline{A}}$ but needs only the first n lags of the autocorrelation and the vector $\underline{b}$. Since the result from solving this linear system is the minimum-phase inverse, the desired wavelet is found by inverting the inverse. This is done in the Fourier domain on line 10. Finally line 11 builds the time-coordinate vector and line 13 normalizes the final wavelet.

Code Snippet 3.5.5. *This code calls* `wavemin` *twice with* $m = 2$ *and* $m = 3$ *to create two different minimum-phase wavelets with the same dominant frequency. There amplitude spectra are also calculate and the result is shown in Figure 3.10*

```
1    %make wavemin wavelet
2    [w1,tw]=wavemin(.002,20,.2,2);
3    [w2,tw]=wavemin(.002,20,.2,3);
4    %compute spectrum
```

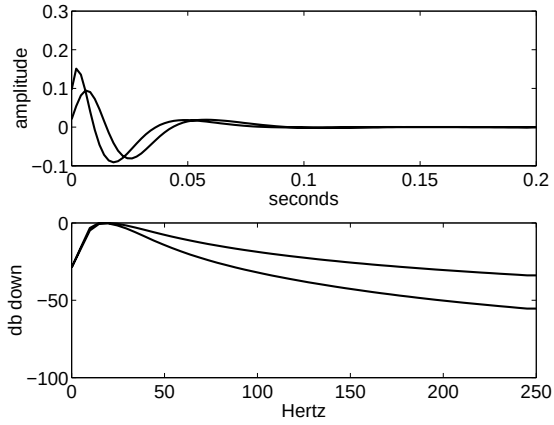

Figure 3.10: Two minimum phase wavelets are shown (top) and their amplitude spectra (bottom). The wavelets were created with Code Snippet 3.5.5 and correspond to $m = 2$ and $m = 3$ in equation (3.49). The $m = 2$ wavelet has a broader spectrum and is more front-loaded in the time domain.

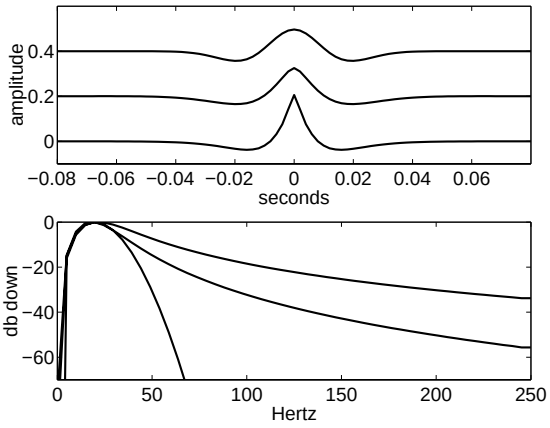

Figure 3.11: In the upper frame, a 20 Hz Ricker wavelet (top) is compared to ***wavez*** wavelets corresponding to $m = 3$ (middle) and $m = 2$ (bottom). Their amplitude spectra are shown in the lower frame. The Ricker spectrum is the one that runs off the bottom of the axes at 70Hz.

```
5     [W1,f]=fftrl(w1,tw);
6     W1=todb(W1);
7     [W2,f]=fftrl(w2,tw);
8     W2=todb(W2);
```

End Code

The example shown in Code Snippet 3.5.5 uses ***wavemin*** to create two different minimum-phase wavelets. The wavelets have the same dominant frequency of 20 Hz and have $m = 2$ and $m = 3$. The wavelet with $m = 2$ shows less high-frequency decay and, in the time domain, is more *front loaded*. It is not certain that the result from ***wavemin*** will be truly minimum phase. If the parameters are chosen such that the requested temporal length is substantially shorter than what is implied by the spectrum, then a non-minimum phase wavelet can result due to truncation effects. The utility function ***ismin*** is provided to test a wavelet for the minimum-phase property. This function factors the z-transform of the wavelet and measures the radius vector for all of the roots. If all roots are outside the unit circle, then ***ismin*** returns unity. If any root is inside the unit circle, ***ismin*** returns 0. The two wavelets in Figure 3.10 pass the minimum phase test; however, `wavemin(.001,10,.2,3)` produces a wavelet that does not.

The spectral model of equation (3.49) can also be used to create a zero-phase wavelet. This is accomplished by ***wavez***. Figure 3.11 shows three zero-phase wavelets with the same dominant frequency of 20 Hz. The upper wavelet is a 20Hz. Ricker while the middle and lower were created by ***wavez*** with $m = 3$ and $m = 2$ respectively. Relative to the Ricker, the ***wavez*** wavelets are considerably more spikelike at the origin. This is due to their relative richness in high frequencies compared with the Ricker wavelet. The code that created this Figure is a simple modification of Code Snippet 3.5.5 and is not shown.

Given any particular amplitude spectrum there are infinitely many wavelets that can be created with different phase spectra. So far, only minimum phase and zero phase have been discussed.

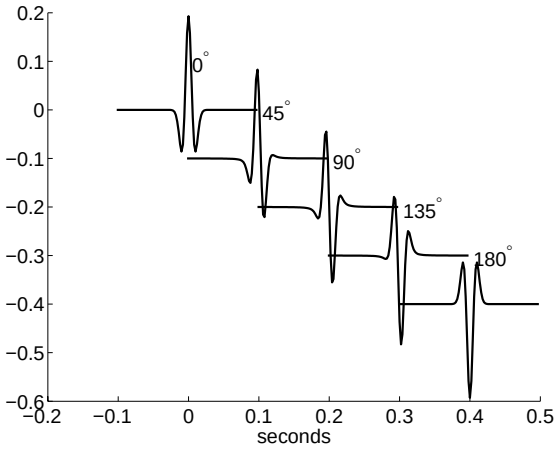

Figure 3.12: A 40 Hz zero-phase Ricker wavelet is shown for four phase rotations of $\theta = [45°, 90°, 135°, 180°]$. This was created with Code Snippet 3.5.6.

Often, it is useful to model seismic data with a wavelet that has a constant phase other than zero. This is easily accomplished by creating a zero-phase wavelet and then using ***phsrot*** to rotate the phase from zero to the desired constant value. An example of this that creates Figure 3.12 is shown in Code Snippet 3.5.6. It is good practice to note the basic features of each phase rotation. For example, the 90° rotation has antisymmetry about the wavelet's center with a major leading peak and a trailing trough. The 45° rotation is similar but the major leading peak is about 50% larger than the trailing trough. Since $e^{i\pi} = -1$, a 180° phase rotation is just a flip in polarity. This can be seen in the 0° and 180° wavelets and it also means that a -45° rotation is the same as a polarity-reversed 135° wavelet. Similarly, the 45° wavelet is a polarity-reversed version of a -135° wavelet.

Code Snippet 3.5.6. *This code creates a zero-phase Ricker wavelet and then calls* `phsrot` *four times to create a set of phase rotated wavelets. The resulting plot is in Figure 3.12.*

```
1    [w,tw]=ricker(.002,40,.2);%make Ricker wavelet
2    nrot=5;deltheta=45;%number of phases and increment
3    figure
4    for k=1:nrot
5        theta=(k-1)*deltheta;
6        wrot=phsrot(w,theta);%phase rotated wavelet
7        xnot=.1*(k-1);ynot=-.1*(k-1);
8        line(tw+xnot,wrot+ynot);%plot each wavelet
9        text(xnot+.005,ynot+.1,[int2str(theta) '^\circ'])
10   end
```

End Code

3.5.2 Seismograms

Convolutional seismograms with random reflectivity

A common use for synthetic seismograms is to test deconvolution algorithms. In this context, it is often desirable to use an artificial reflectivity function so that the deconvolution assumption of a

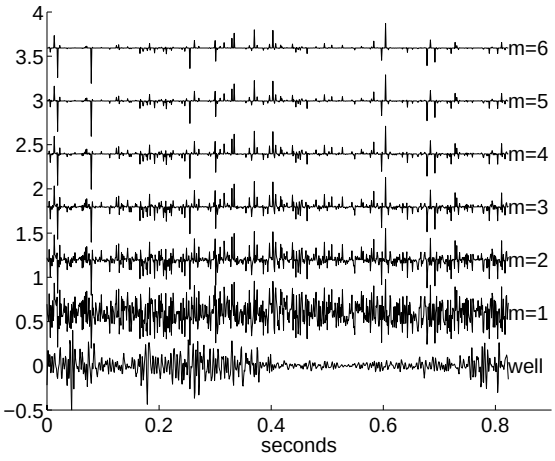

Figure 3.13: Seven reflectivity functions are shown. The bottom one comes from a real well log and the other six are synthetic reflectivities that were generated by ***reflec***.

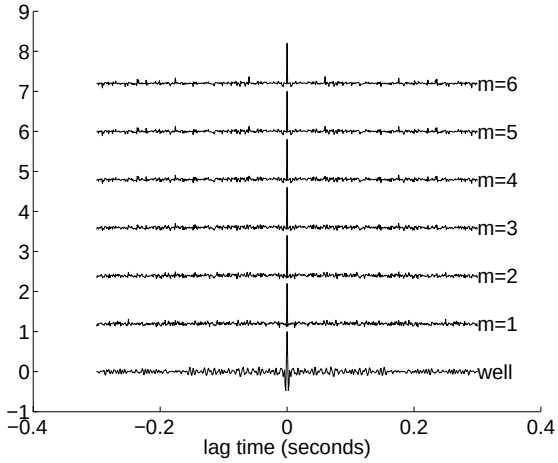

Figure 3.14: Seven two-sided autocorrelations are shown corresponding to the seven reflectivities in Figure 3.13.

white reflectivity is fulfilled as nearly as possible. The command ***reflec*** supplies such a reflectivity function by using MATLAB's random number generator, `randn`. Technically, numerical random number generators like `randn` generate *pseudo-random* numbers. This means that given the same starting point, called the *seed*, the generator will return the same set of apparently random numbers. MATLAB has two random number generators, `rand` that generates *uniformly distributed* random numbers, and `randn` that generates *normally, or Gaussian, distributed* random numbers. Generally, a normal distribution is preferable, however, in either case such random numbers do not display the *spikiness* typically found in real well logs. This is addressed in ***reflec*** by raising the output from `randn` to an integral power, m, as in

$$r_{reflec} = sign(r_{randn}) \, |r_{randn}|^m \, . \tag{3.50}$$

Suppose that the maximum values in r_{randn} are near unity. Then raising them to any power will not change them while the smaller numbers will get smaller. Thus r_{reflec} will appear more spiky than r_{randn}.

Figure 3.13 compares a real reflectivity derived from well logs with six outputs from ***reflec*** using values of of m from 1 to 6. It is a subjective decision about which value of m gives the most realistic reflectivity estimate. The default in ***reflec*** is 3. Clearly, the real reflectivity has features that are present in none of the synthetic ones. The most obvious is the low-amplitude zone from .4 to .7 seconds on the real reflectivity. More generally, a low frequency amplitude variation can be perceived throughout the real reflectivity while the synthetic ones show a very uniform amplitude distribution with time. This is a hint that the real reflectivity is not white, i.e. not truly random, but rather shows *spectral color*.

A useful test of randomness is to calculate the autocorrelation of the reflectivity. In theory, a truly random signal has an autocorrelation that is exactly zero everywhere except at zero lag where it is unity. However, this requires an infinite length signal and so is never exactly fulfilled in practice. Figure 3.14 shows the central portion of the autocorrelations of the reflectivities of Figure 3.13. All

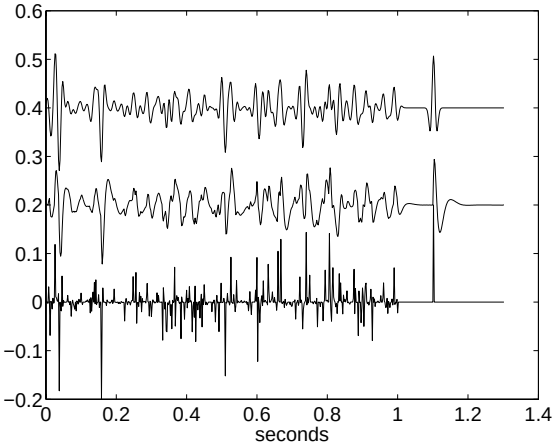

Figure 3.15: The lower trace is a random reflectivity generated by *reflec* with an extra spike on the end. The middle trace is a convolutional seismogram that uses a 20 Hz minimum phase wavelet convolved with the first trace. The top trace is a similar seismogram except that a 20 Hz minimum-phase wavelet was used. See Code Snippet 3.5.7.

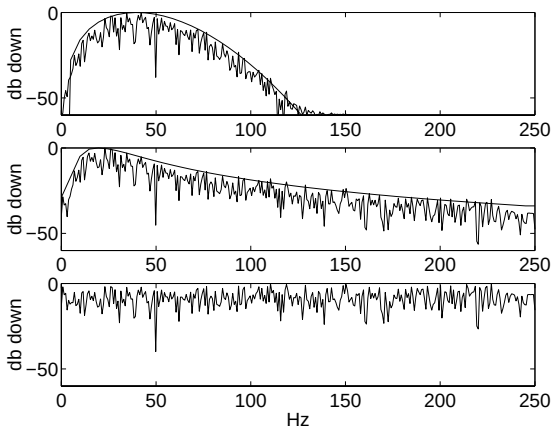

Figure 3.16: Decibel amplitude spectra of the seismograms of Figure 3.15: (bottom) reflectivity, (middle) minimum-phase seismogram and wavelet (smooth curve), and (top) Ricker seismogram and wavelet (smooth curve).

autocorrelations have a large spike at zero lag and only small values elsewhere. However, only the real well log shows significant negative side lobes of the zero lag sample. This again indicates that the real reflectivity is not white but has spectral color. It is also interesting that r_{reflec} shows the same degree of randomness regardless of the value of m.

Code Snippet 3.5.7. *Here two wavelets, a 40 Hz Ricker and a 20 Hz minimum phase, are convolved with the same reflectivity to generate two synthetic seismograms. The result is shown in Figure 3.15.*

```
1    %make wavelet
2    dt=.002;tmax=1;tmaxw=.2;
3    [wm,twm]=wavemin(dt,20,tmaxw); %20 Hz min phs
4    [wr,twr]=ricker(dt,40,tmaxw); %40 Hz Ricker
5    %make reflectivity
6    [r,t]=reflec(tmax,dt,.2);
7    %pad spike at end
8    ntw=tmaxw/dt;
9    r=[r;zeros(ntw/2,1);.2;zeros(ntw,1)];
10   t=dt*(0:length(r)-1);
11   %convolve and balance
12   s1=convz(r,wr);
13   s1=balans(s1,r);
14   s2=convm(r,wm);
15   s2=balans(s2,r);
```

End Code

The calculation of a synthetic seismogram using *reflec* to generate the reflectivity is illustrated in Code Snippet 3.5.7. Lines 3-4 generate a 40 Hz Ricker wavelet and a 20 Hz minimum-phase wavelet

as described in section 3.5.1. Line 6 invokes *reflec* to create a random reflectivity. The three input parameters are the maximum time, the time sample rate, and the maximum reflection coefficient. A fourth argument, that is defaulted here, gives m in equation (3.50). In addition to returning a reflectivity series, `r`, *reflec* also returns an appropriate time-coordinate vector, `t`. Lines 8-10 place an extra isolated spike on the end of the reflectivity. This is useful because, after convolution, the shape of the wavelet will be immediately apparent. Line 9 is simply the concatenation of the reflectivity with: a column vector of `ntw/2` zeros, an impulse of .2, and another column vector of `ntw` zeros. Line 10 rebuilds the time-coordinate vector to be compatible with the lengthened reflectivity. Line 12 convolves the reflectivity with the zero-phase Ricker wavelet using the convenience function *convz* while line 14 convolves the reflectivity with the minimum-phase wavelet using *convm*. These convenience functions invoke `conv` and then truncate the result to be the same length as the first input argument. Function *convz* truncates evenly from the beginning and the end of the convolved series as is appropriate if a zero-phase wavelet was used. Function *convm* truncates entirely off the end of the convolved series as would be expected for a causal wavelet. After each convolution, the function *balans* is invoked to balance the amplitudes of the seismogram with the original reflectivity to aid in the comparison. One thing that is immediately apparent from these seismograms is that the zero-phase seismogram is more *interpretable* than the minimum-phase. This is because the larger peaks on the former coincide temporally with the major reflection coefficients. With the minimum-phase seismogram, the major peaks are displaced to later times relative to the reflectivity.

Code Snippet 3.5.8. *This example assumes that Code Snippet 3.5.8 has already been run. Here the spectra are computed and displayed. The result is Figure 3.16.*

```
1    [R,f]=fftrl(r,t);
2    [S1,f]=fftrl(s1,t);
3    [S2,f]=fftrl(s2,t);
4    [Wr,fwr]=fftrl(wr,twr);
5    [Wm,fwm]=fftrl(wm,twm);
6    R=real(todb(R));
7    S1=real(todb(S1));
8    S2=real(todb(S2));
9    Wr=real(todb(Wr));
10   Wm=real(todb(Wm));
```

End Code

Code Snippet 3.5.8 assumes that Code Snippet 3.5.7 has already been run. Lines 1-5 use *fftrl* to compute the spectrum of each component of the seismogram construction exercise. Then, lines 6-10 use the convenience function *todb* to compute the amplitude spectrum in decibel format, each spectrum relative to its own maximum. (The imaginary part of the return from *todb* is the phase spectrum). The resulting spectra are displayed in Figure 3.16. It is apparent that the spectra of the seismograms are shaped by that of the wavelets. That is, the slowly-varying, average behavior of the seismogram spectrum is essentially similar to the wavelet spectrum. The fact that the Ricker wavelet spectrum departs from that of the seismogram in the upper panel is an artifact caused by the truncation of the convolution.

It is often desirable to add some level of background random noise to a synthetic to simulate a noisy recording environment or perhaps the limited precision of recording. The function *rnoise* is designed for this purpose. If `s1` is a time series, then `rn=rnoise(s1,5)` creates a vector of normally-distributed random noise, `rn`, such that `s1+rn` will have a signal-to-noise ratio of 5. The signal-to-noise ratio is defined by assuming that `s1` is pure signal and measuring its rms amplitude as $a_{rms} = \sqrt{\sum_j s1_j^2}$, where the sum is over samples in the time domain. The rms amplitude of `rn`

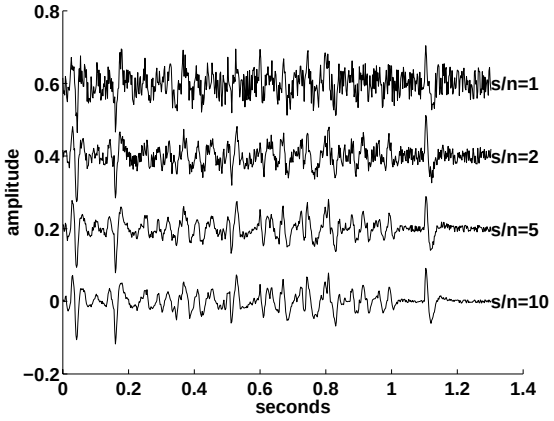

Figure 3.17: The minimum-phase seismogram of Figure 3.15 is shown with four different levels of added noise.

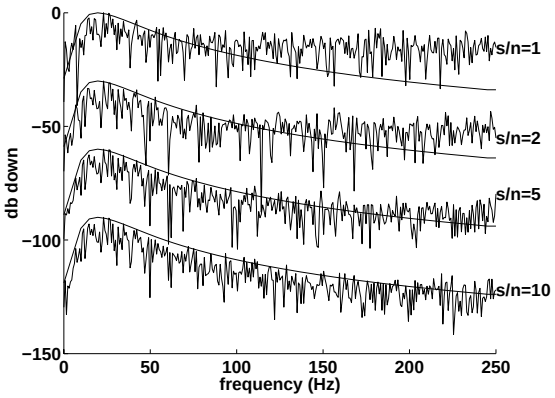

Figure 3.18: The spectra of the seismograms of Figure 3.17 are shown with the spectrum of the wavelet superimposed on each. The maxima of the spectra acutally coincide but have been shifted vertically for clarity.

is then set as σa_{rms} where σ is the desired signal-to-noise ratio. In using ***rnoise*** it is important to remember that `rn` must be added to the input time series after calling ***rnoise***. Also, the method of calculating the signal-to-noise ratio may not be exactly what was desired as there is no standard way of doing this. Other possibilities include defining the ratio between peak amplitudes or in the frequency domain. Also, ***rnoise*** can be given a third argument that defines the range of indices over which the signal-to-noise ratio is to be computed. This is useful for nonstationary signals whose signal level may change strongly with time. For example, if `t` is the time coordinate vector for `s1`, then `rn=rnoise(s1,5,near(t,.5,1.0))` uses the function ***near*** to determine the range of indices that span the time zone from .5 to 1.0 seconds. Then only those samples between .5 and 1 are used in the rms amplitude calculation. Figure 3.17 shows four different noisy seismograms computed using ***rnoise*** and the minimum-phase seismogram of Figure 3.15 as signal. The Fourier amplitude spectra of these noisy seismograms are shown in Figure 3.18 with the spectrum of the wavelet superimposed. Comparison with Figure 3.16 shows that the noise causes a *corner* in the spectrum defined by where it departs from the shape of the wavelet's spectrum. An interpretive assessment of the corner frequencies would put them near 75 Hz, 100 Hz, 150 Hz, and 220 Hz for signal-to-noise ratios of 1, 2, 5, and 10 respectively. It is very difficult to recover the signal spectrum beyond this point in data processing. Close inspection shows that the spectra at frequencies below the corner frequency are also altered by the addition of noise. From another perspective, this must be the case because the dynamic range of the wiggle trace plots in Figure 3.17 is about 20-30 decibels and they are all clearly altered by the noise.

Exercise 3.5.1. *Develop a script that remakes and displays the seismograms and spectra of Figures 3.15 and 3.16. Modify the script to demonstrate that truncation effects are the reason that the seismogram spectrum and the Ricker wavelet spectra depart from one another near 110 Hz (top Figure 3.16.*

Exercise 3.5.2. *Create a script that builds two seismograms from the same reflectivity using wavelets from* `wavemin` *and* `wavez`*. Make the wavelets the same in all respects except for their phase. Which*

seismogram is more interpretable? Why?

Synthetic seismograms with multiples

The

Chapter 4

Velocity

Exploration seismology is literally overflowing with *velocities*. Just to name a few, there are interval velocity, instantaneous velocity, apparent velocity, rms velocity, average velocity, mean velocity, stacking velocity, horizontal velocity, vertical velocity, phase velocity, group velocity, P-wave velocity, S-wave velocity, migration velocity, weathering velocity, and almost certainly others. This chapter is meant to bring some order to this chaos of velocities. For starters, there is a fundamental distinction between *physical velocities* and *velocity measures*. The former refers to velocities that are actually the speed at which some physical wave propagates. Examples are instantaneous velocity, P- and S-wave velocities, phase and group velocity. On the other hand, velocity measures are typically quantities derived from data analysis that have the physical dimensions of velocity but are related to physical velocities in some indirect (and possibly unknown) fashion. Examples of velocity measures include average, mean, and rms velocities, interval velocity, stacking velocity, apparent velocity, and migration velocity. In contrast to physical velocities, it cannot generally be expected that a physical wave actually propagates at the speed of one of these velocity measures.

Using the measured data for the analysis of velocity and the application to the data of corrections that depend upon velocity are fundamental to seismic processing. Velocity, together with the geometry of the seismic acquisition and the shapes of the reflectors, causes the characteristic traveltime shapes (such as hyperbolae) in the data. Spatial variations in velocity cause distortions from the simple, canonical shapes, and these distortions must be accounted for in processing. Sometimes velocity information can be obtained from supporting data such as well logs, core measurements, and geologic maps but this information is inherently sparse with respect to the coverage of the seismic data itself. Ultimately, the only way to obtain the spatially dense velocity information required in processing is from the data itself. This is known as *velocity analysis* and is the source of some of the velocity measures such as stacking and migration velocities. The interpretation of the velocity measures such that they can be converted into measurements of physical velocities is called *velocity inversion* and is an ongoing problem. A careful discussion of the definitions of these measures is essential to gain understanding of the complexities of velocity inversion.

For simplicity, consider the case of P-wave propagation in a heterogeneous earth. (S-waves can be treated similarly.) A velocity measure will be defined by either relating it mathematically to the actual P-wave speed or by describing how it can be derived from seismic data.

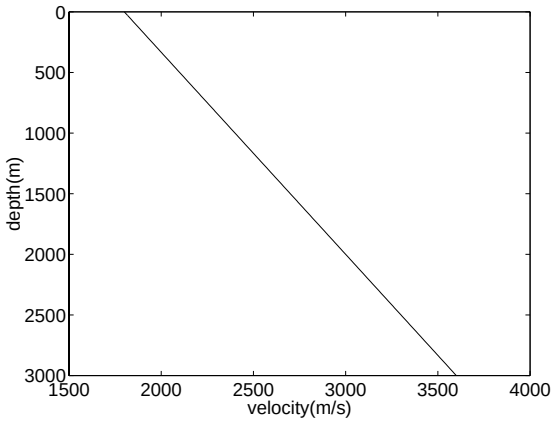

Figure 4.1: The universal velocity function is linear with depth

4.1 Instantaneous velocity: v_{ins} or just v

Instantaneous velocity generally refers to the speed of propagation of seismic waves in the earth. The word *instantaneous* refers to the local wave speed as the wave evolves from one instant of time to the next. The term can be applied to any wave type (P, S, surface, etc) though here we specialize to P-waves for clarity.

Like most seismic "velocities" v_{ins} is not usually a vector quantity and so is not a velocity as physicists would use the term. Rather it is a scalar which can be thought of as the magnitude of a velocity vector.

For practical seismic experiments, v_{ins} must be acknowledged to be a highly variable function of position in the earth. In the most general case of anisotropic media, it also depends upon direction, but this will be ignored for now. It is not a function of time as that would imply that the earth's properties changed during the seismic experiment. Only rarely can it be assumed to be spatially constant but often there is significant variation in one direction only. This is commonly the case in sedimentary basins where velocity varies strongly with depth but only weakly in the horizontal direction.

Consider the instantaneous velocity function that is linear with depth z

$$v_{ins} = v_0 + cz. \tag{4.1}$$

The *universal velocity function* is the name given to this function when $v_0 = 1800$ m/sec and $c = .6$ sec^{-1}. Figure 4.1 shows this function for a depth range of 0 to 3000 m. The name originates in the days of analog computing (the 1950s and 1960s) when changing a velocity function was a considerable labor. For example, *normal moveout* was removed by a "computer" (who was a human being) who "traced the trace" (hence the name *trace* for a seismic recording) with a stylus that was connected to a pen by a lever arm driven by an eccentric cam. The pen would redraw the trace with normal moveout removed. However, the shape of the cam was determined by the velocity function and changing functions often meant going to the machine shop to cut a new cam. Thus the possibility of a universal function was quite attractive. And, it actually works quite well in many sedimentary basins.

4.2 Vertical traveltime: τ

Given a complete specification of v_{ins} as a function of position, $v_{ins}(x, y, z)$, traveltimes over any path can be computed. Since there are infinitely many possible paths between any two points (of course, not all are Snell's law paths but all are possible routes for scattered energy), it is unreasonable to expect that there could be an unambiguous relation between traveltime and depth. However, if a special path is chosen, then such a relation can be developed. For this purpose, it is common to relate depth, z, to the traveltime along the vertical path from 0 to z. This *vertical traveltime* is a useful quantity that, to first order, is the time coordinate of final *stacked sections* and, to higher accuracy, is the time coordinate for *migrated time sections*. Even though v_{ins} is a general function of position, for the calculation of vertical traveltime, it can be considered as a function of depth alone because the calculation at each (x, y) is independent. Therefore, instantaneous velocity will now be written $v_{ins}(z)$ with the understanding that the calculations can be done for any and all (x, y).

Vertical traveltime is calculated by considering a small depth interval, dz over which v_{ins}can be considered to be approximately constant. The vertical traveltime over this interval is simply

$$d\tau = \frac{dz}{v_{ins}(z)}. \tag{4.2}$$

The total one-way traveltime from the surface to depth, z, is simply the sum of many such small contributions. In the limit as $dz \to 0$ this becomes the integral

$$\tau(z) = \int_0^z \frac{d\tilde{z}}{v_{ins}(\tilde{z})}. \tag{4.3}$$

The z dependence appears as the upper limit of the integral while $\tilde{z}$ is just a dummy variable of integration. Defined in this way, $\tau(z)$ is a function that increases monotonically with z. As such it is guaranteed to have an inverse. That is, given $\tau(z)$ it is always possible to find $z(\tau)$ and vice-versa. Function $\tau(z)$ is called a *time-depth curve* and can be used to find depth given vertical traveltime or the reverse.

Continuing with the universal velocity function of Equation 4.1, the vertical traveltime of equation (4.3) becomes

$$\tau(z) = \int_0^z \frac{d\tilde{z}}{v(\tilde{z})} = \frac{1}{c}\int_{v_0}^{v_0+cz} \frac{d\xi}{\xi} = \frac{1}{c}\ln\left[1 + \frac{cz}{v_0}\right]. \tag{4.4}$$

Thus a linear variation of velocity with depth yields a logarithmic time-depth curve. It is a simple matter to invert Equation 4.4 for $z(\tau)$ to get

$$z(\tau) = \frac{v_0}{c}\left[e^{c\tau} - 1\right]. \tag{4.5}$$

For the case of the universal velocity function, the curves $\tau(z)$ and $z(\tau)$ are shown in Figures 4.2 and 4.3 respectively. The graph of $z(\tau)$ is just the transpose of the graph of $\tau(z)$.

4.3 v_{ins} as a function of vertical traveltime: $v_{ins}(\tau)$

In the preceding sections it has been shown that for any fixed (x, y) location, every depth, z, has a unique vertical traveltime associated with it. Therefore, v_{ins}, which is physically a function of z

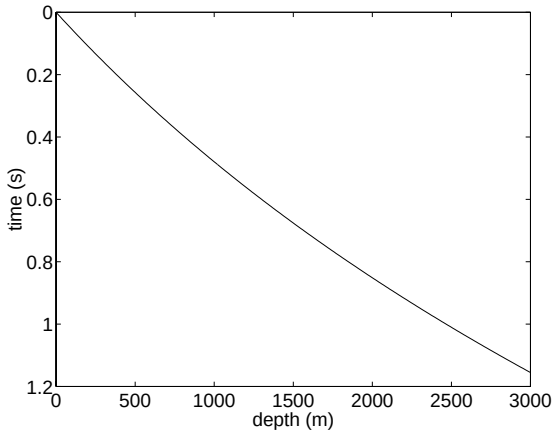

Figure 4.2: The time-depth curve ($\tau(z)$) of equation (4.4) for the case of $v_0 = 1800$m/sec and $c = .6$/sec

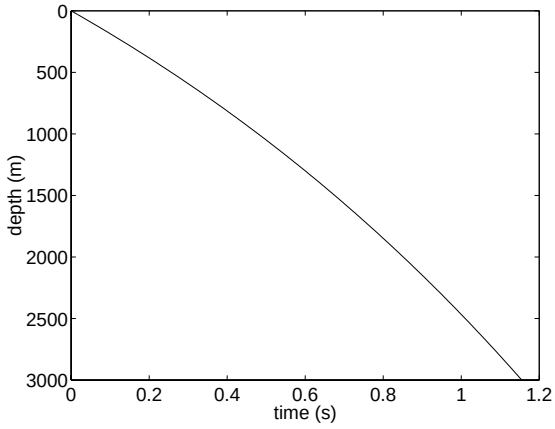

Figure 4.3: The depth-time curve ($z(\tau)$) of equation (4.4) for the case of $v_0 = 1800$ m/sec and $c = .6$ sec^{-1}

may always be expressed as a function of $\tau(z)$. That is

$$v_{ins}(\tau) = v_{ins}(z(\tau)) \tag{4.6}$$

Given $v_{ins}(\tau)$ in equation (4.2), an expression for $z(\tau)$ follows as

$$z(\tau) = \int_0^\tau v_{ins}(\tilde{\tau})d\tilde{\tau}. \tag{4.7}$$

Comparing equations (4.3) and (4.7) shows that knowledge of $v_{ins}(z)$ allows computation of $\tau(z)$ and knowing $v_{ins}(\tau)$ enables computation of $z(\tau)$. In practice, $v_{ins}(z)$ might be directly obtained from a sonic well log while $v_{ins}(\tau)$ can be estimated from the analysis of stacking velocities.

For the universal velocity function of equation (4.1) the vertical traveltime results in the logarithmic expression given in equation (4.4) and $z(\tau)$ is given by equation (4.5). Thus $v_{ins}(\tau)$ becomes

$$v_{ins}(\tau) = v_{ins}(z(\tau)) = v_0 + c\left[\frac{v_0}{c}\left(e^{c\tau} - 1\right)\right] \tag{4.8}$$

which reduces to

$$v_{ins}(\tau) = v_0 e^{c\tau}. \tag{4.9}$$

Thus, when v_{ins} is linear with depth it is actually exponential with vertical traveltime. For case of the universal velocity function, $v_{ins}(\tau)$ is plotted in Figure 4.4.

4.4 Average velocity: v_{ave}

The typical industry definition of v_{ave} is that it is a particular depth divided by the vertical traveltime from the surface to that depth. Given any $z(\tau)$ curve, pick a point (τ_0, z_0) and v_{ave} is the slope of the line connecting the origin with the point (τ_0, z_0) while v_{ins} is the tangent to the curve at that

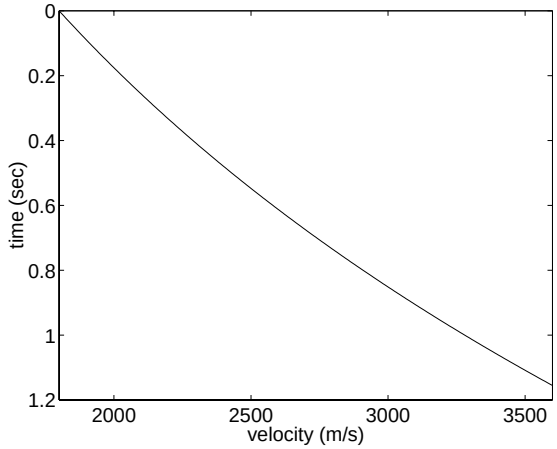

Figure 4.4: For the universal velocity function shown in Figure 4.1, v_{ins} is an exponential function of vertical traveltime.

point. Mathematically, $v_{ave}(z)$ is expressed as

$$v_{ave}(z) = \frac{z}{\tau(z)} = \frac{z}{\int_0^z \frac{d\tilde{z}}{v_{ins}(\tilde{z})}} \tag{4.10}$$

while $v_{ave}(\tau)$ is

$$v_{ave}(\tau) = \frac{z(\tau)}{\tau} = \frac{1}{\tau}\int_0^\tau v_{ins}(\tilde{\tau})d\tilde{\tau}. \tag{4.11}$$

Equation (4.11) shows that the average velocity is a mathematical average only with respect to vertical traveltime. (Recall from calculus, that the average (or mean) value of a function $f(x)$ over the interval from $0 \to x$ is $x^{-1}\int_0^x f(x')dx'$.) When considered as a function of depth, the average velocity is not a mathematical average.

For the running example of linear variation of velocity with depth, $v_{ave}(z)$ becomes

$$v_{ave}(z) = \frac{cz}{\ln\left[1 + \frac{cz}{v_0}\right]} \tag{4.12}$$

and $v_{ave}(\tau)$ is the given by

$$v_{ave}(\tau) = \frac{v_0}{c\tau}\left[e^{c\tau} - 1\right]. \tag{4.13}$$

These equations are graphed in Figures 4.5 and 4.6.

4.5 Mean velocity: v_{mean}

Since v_{ave} is a mathematical average of v_{ins} over traveltime and not depth, it is useful to define another velocity measure that is a depth average. The mean velocity, v_{mean} is defined as

$$v_{mean}(z) = \frac{1}{z}\int_0^z v_{ins}(\tilde{z})d\tilde{z}. \tag{4.14}$$

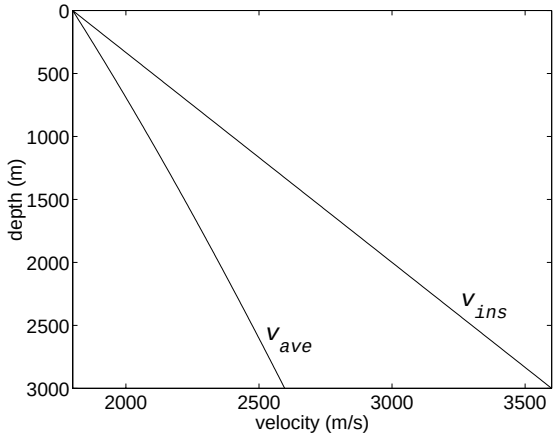

Figure 4.5: v_{ave} and v_{ins} are shown versus depth for the case of the universal velocity functions (equation (4.1)) with v_0 = 1800 m/s and c = .6 sec^{-1}

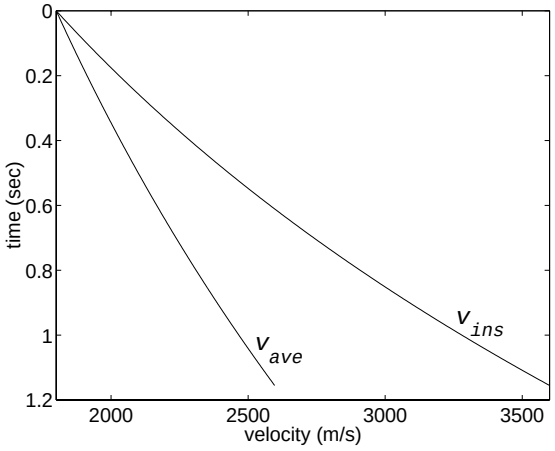

Figure 4.6: v_{ave} and v_{ins} are shown versus time for the case of the universal velocity functions (equation (4.1)) with v_0 = 1800 m/s and c = .6 sec^{-1}

For the linear variation of $v_{ins}(z)$, $v_{mean}(z)$ becomes

$$v_{mean}(z) = \frac{1}{z}\int_0^z [v_0 + c\tilde{z}]\, d\tilde{z} = v_0 + \frac{1}{2}cz \tag{4.15}$$

which shows that the mean velocity increases at half the rate of $v_{ins}(z)$. Of course, $v_{mean}(z)$ can be re-expressed as a function of τ by substituting $z(\tau)$, this results in

$$v_{mean}(\tau) = \frac{v_0}{2}\left[1 + e^{c\tau}\right]. \tag{4.16}$$

Equation (4.15) is compared with $v_{ave}(z)$ and $v_{ins}(z)$ in Figure 4.7. $v_{mean}(z)$ and $v_{ins}(z)$ are both linear but $v_{ave}(z)$ is not. Also, $v_{mean}(z)$ is always greater than $v_{ave}(z)$ which will be proven true in the next section.

4.6 RMS velocity: v_{rms}

The mean velocity, discussed previously, is not commonly used in seismology. The reason is that another velocity measure, v_{rms}, is used and only two of v_{ave}, v_{mean}, and v_{rms} are independent. The root-mean-square velocity is defined as

$$v_{rms}^2(\tau) = \frac{1}{\tau}\int_0^\tau v_{ins}^2(\tilde{\tau})d\tilde{\tau}. \tag{4.17}$$

Two important relationships exist between the three velocity measures: v_{ave}, v_{mean}, and v_{rms}. First, they are linked by the relation

$$v_{rms}^2 = v_{ave}v_{mean} \tag{4.18}$$

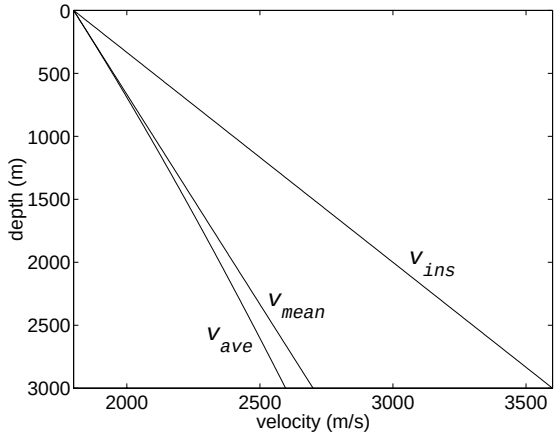

Figure 4.7: v_{mean} and v_{ave} are compared with v_{ins} for the case of the linear increase of v_{ins} with z.

which means that only two of the three velocity measures can be considered independent. The proof is straight forward $v_{rms}^2 = \tau^{-1} \int v_{ins}^2 d\tau = \frac{z}{\tau}\frac{1}{z} \int v_{ins}[v_{ins} d\tau] = \frac{z}{\tau}[\frac{1}{z} \int v_{ins} dz] = v_{ave} v_{mean}$.

The second relationship follows from a mathematical inequality known as *Schwartz's Inequality*. This result is quite general and confirms the intuition that the square root of the average of the squares of a set of numbers is always greater than the direct average of the numbers themselves. That is, let $\{x_k\}$ be an arbitrary set of n real numbers, then $\sqrt{n^{-1} \sum_k x_k^2} \geq n^{-1} \sum_k x_k$ where the equality only occurs if all x_k are identical.

For the continuing example of v_{ins} linear with depth, equation (4.9) showed that v_{int} was exponential. Thus v_{rms} becomes

$$v_{rms}^2(\tau) = \frac{1}{\tau} \int_0^\tau v_0^2 e^{2c\tilde{\tau}} d\tilde{\tau} = \frac{v_0^2}{2c\tau} \left[e^{2c\tau} - 1 \right]. \tag{4.19}$$

Figure 4.8 compares v_{rms} v_{ave} and v_{ins} for the universal velocity function. Though v_{rms} is always greater than v_{ave} the difference is not greater than 2%.

Exercise 4.6.1. *Use MATLAB to numerically demonstrate Schwartz's inequality. Use* `rand` *to generate a vector of random numbers with zero mean. Compute both the mean and the rms average of these numbers. Repeat the process for many different vectors with steadily increasing length. Plot a graph of these two averages versus length. Repeat your work using random numbers that are all positive with a mean value somewhere near that expected for seismic velocities. Show that if the velocity vector has identical entries then the rms average and the mean are equal but if one value differs from the rest, then the rms average exceeds the mean.*

4.7 Interval velocity: v_{int}

Corresponding to any of the velocity averages, an *interval velocity* can be defined that is simply the particular average applied across a small interval rather than from the surface ($z = 0$) to depth z.

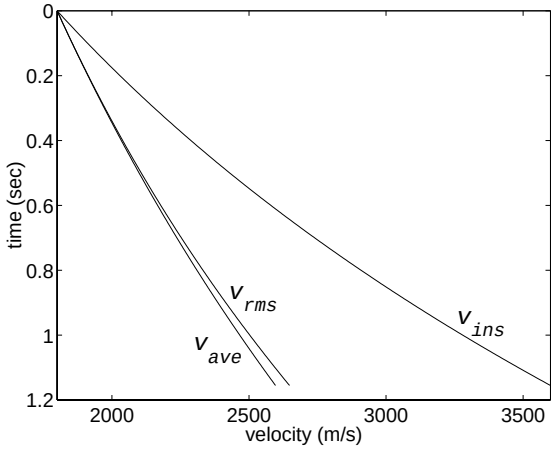

Figure 4.8: v_{rms} and v_{ave} are compared with v_{ins} for the case of the linear increase of v_{ins} with z.

For example, the average and rms velocities across an interval defined by τ_1 and τ_2 are simply

$$v_{ave}\left(\tau_2, \tau_1\right) = \frac{1}{\tau_2 - \tau_1} \int_{\tau_1}^{\tau_2} v_{ins}(\tilde{\tau}) d\tilde{\tau} \tag{4.20}$$

and

$$v_{rms}^2\left(\tau_2, \tau_1\right) = \frac{1}{\tau_2 - \tau_1} \int_{\tau_1}^{\tau_2} v_{ins}^2(\tilde{\tau}) d\tilde{\tau}. \tag{4.21}$$

These quantities are functions of both the upper and lower bounds of the integrals. In the limit, as the interval shrinks so small that $v_{ins}(\tau)$can be considered constant, both interval velocities approach the instantaneous velocity.

It follows directly from the definition of average velocity (equation (4.10)) that the average velocity across a depth interval is just the ratio of the depth interval to the vertical traveltime across the interval. That is

$$v_{ave}\left(\tau_2, \tau_1\right) = \frac{z_2 - z_1}{\tau_2 - \tau_1} = \frac{\Delta z}{\Delta \tau}. \tag{4.22}$$

Thus, if the range from 0 to z is divided into n finite intervals defined by $z_0, z_1, z_2, \ldots, z_{n-1}, z_n$ (where $z_0 = 0$ and $z_n = z$) then

$$v_{ave}(\tau) = \frac{z(\tau)}{\tau} = \frac{\sum_{k=1}^{n} \left[z_k - z_{k-1}\right]}{\tau} = \frac{1}{\tau} \sum_{k=1}^{n} [\tau_k - \tau_{k-1}] v_{ave}(\tau_k, \tau_{k-1}) \tag{4.23}$$

where $\tau_k = \tau(z_k)$. Defining $\Delta\tau_k = \tau_k - \tau_{k-1}$ then gives

$$v_{ave}(\tau) = \frac{1}{\tau} \sum_{k=1}^{n} v_k \Delta\tau_k \tag{4.24}$$

where $v_k \equiv v_{ave}(\tau_k, \tau_{k-1})$ and $\tau = \sum_{k=1}^{n} \Delta\tau_k$. Comparing equation (4.24) with equation (4.11) suggests that the former is just a discrete version of the latter. However, the velocity v_k in equation

(4.24) is the average velocity of the k$^{\text{th}}$ finite interval and is thus the time average of v_{ins} across the interval. Of course, if $v_{ins}(\tau)$ is constant across each interval then v_k is an instantaneous velocity and this distinction vanishes.

Equation (4.24) can be used in a number of ways. Most obviously, it shows how to combine a set of *local average velocities* to obtain the *macro average velocity* across a larger interval. Also, it can be used to estimate a local average velocity given two macro-average velocities. Suppose the average velocities v_{ave1} and v_{ave2} from $z = 0$ to depths z_1 and z_2 are known. Then an expression for the average velocity, from $z_1 \to z_2$ or equivalently from $\tau_1 \to \tau_2$, follows from equation (4.24) and is

$$v_{ave}(\tau_2, \tau_1) = \frac{1}{\tau_2 - \tau_1} \left[\tau_2 v_{ave2} - \tau_1 v_{ave1}\right]. \tag{4.25}$$

The two macro averages are each weighted by their time intervals and then the shallower average is subtracted from the deeper. This difference is then divided by the time interval of the local average.

If $v_{ave1}, \tau_1, v_{ave2}, \text{and} \tau_2$ are all measured without error, then this process (i.e. equation 4.25) works perfectly. However, in a real case, errors in the measurements can lead to wildly incorrect estimates of $v_{ave}(\tau_2, \tau_1)$. With noisy data, there is no guarantee that the term $[\tau_2 v_{ave2} - \tau_1 v_{ave1}]$ will always be positive leading to the possibility of negative velocity estimates. Also, the division by $\tau_2 - \tau_1$ can be unstable if the two times are very close together.

The discussion so far has been only for average velocities across an interval. The entire derivation above can be repeated for rms velocities with similar results but a few more subtleties arise in interpretation. Rather than repeating the derivation just given with 'rms' in place of 'ave', it is instructive to consider an alternative approach. It is a property of integrals that $\int_a^c = \int_a^b + \int_b^c$ where $a < b < c$. Given $\tau_0 < \tau_1 < \tau_2$ and applying this rule to equation (4.17) results in

$$v_{rms}^2(\tau_2, \tau_0) = \frac{1}{\tau_2 - \tau_0} \left[\int_{\tau_0}^{\tau_1} v_{ins}^2(\tilde{\tau}) d\tilde{\tau} + \int_{\tau_1}^{\tau_2} v_{ins}^2(\tilde{\tau}) d\tilde{\tau}\right]. \tag{4.26}$$

Recognizing the integrals in $[\ \ldots\ ]$ as rms interval velocities squared multiplied by their interval times (i.e.$\int_{\tau_0}^{\tau_1} v_{ins}^2(\tilde{\tau}) d\tilde{\tau} = [\tau_1 - \tau_0] v_{rms}^2(\tau_1, \tau_0)$ and similarly for the other integral) leads to

$$v_{rms}^2(\tau_2, \tau_0) = \frac{1}{\tau_2 - \tau_0} \left[(\tau_1 - \tau_0) v_{rms}^2(\tau_1, \tau_0) + (\tau_2 - \tau_1) v_{rms}^2(\tau_2, \tau_1)\right]. \tag{4.27}$$

For the case of n subdivisions between τ_2 and τ_0 this generalizes to

$$v_{rms}^2(\tau_2, \tau_0) = \frac{1}{\tau_2 - \tau_0} \sum_{k=1}^{n} v_k^2 \Delta\tau_k \tag{4.28}$$

where $v_k = v_{rms}(\tau_k, \tau_{k-1})$ and, as before, $\Delta\tau_k = \tau_k - \tau_{k-1}$ and $\tau = \sum_{k=1}^{n} \Delta\tau_k$. This is the rms equivalent to equation (4.24) and all of the comments made previously about v_{ave} apply here for v_{rms}. In particular, equation (4.28) should be thought of a combining interval rms velocities into a macro rms velocity. Only in the case when v_{ins} does not vary significantly across an interval can the v_k in equation (4.28) be considered to be instantaneous velocities.

Equation (4.27) is the *addition rule* for rms velocities. To combine rms velocities, the squared velocities are added and each must be weighted by its time interval. Equation (4.27) can be rearranged

to give an expression for estimating a local rms velocity from two macro velocities

$$v_{rms}^2(\tau_2,\tau_1) = \frac{1}{\tau_2-\tau_1}\left[(\tau_2-\tau_0)v_{rms}^2(\tau_2,\tau_0) - (\tau_1-\tau_0)v_{rms}^2(\tau_1,\tau_0)\right] \tag{4.29}$$

or, with simplified notation

$$v_{rms}^2(\tau_2,\tau_1) = \frac{1}{\tau_2-\tau_1}\left[\tau_2 v_{rms2}^2 - \tau_1 v_{rms1}^2\right]. \tag{4.30}$$

In this expression τ_0 has been set to 0 and $v_{rms2} = v_{rms}(\tau_2,\tau_0)$ and similarly for v_{rms1}. Equation (4.30) is often called the *Dix equation for interval rms velocities* because it was C.H. Dix (Dix, 1955) who first recognized the connection between stacking velocities and rms velocities and showed how to calculate an interval velocity from two stacking velocity measurements.

The application of equation (4.30) in practice requires the measurement of stacking velocities and vertical traveltimes to two closely spaced reflectors. Under the assumption that stacking velocities are well approximated by rms velocities (Dix (1955) or Taner and Koehler (1969)), the rms velocity of the interval is then estimated with equation (4.30). However, as with average velocities, errors in measurement lead to problems with the estimation of $v_{rms}(\tau_2,\tau_1)$. If the term $\left[\tau_2 v_{rms2}^2 - \tau_1 v_{rms1}^2\right]$ becomes negative then imaginary interval velocity estimates result. Thus one essential condition for the use of this technique is that

$$v_{rms2}^2 > v_{rms1}^2\frac{\tau_1}{\tau_2}. \tag{4.31}$$

Since $\tau_1/\tau_2 < 1$, this is a constraint upon how fast v_{rms} estimates can decrease with increasing time. There is no mathematical basis to constrain the rate at which v_{rms} can increase; however, it is reasonable to formulate a constraint on physical grounds. Since P-wave seismic velocities are not expected to exceed some v_{max} (say 7000 m/s) a constraint would be

$$v_{rms2}^2 < v_{rms1}^2\frac{\tau_1}{\tau_2} + v_{max}^2\frac{\tau_2-\tau_1}{\tau_2}. \tag{4.32}$$

Exercise 4.7.1. *Show that the right-hand-side of inequality* (4.32) *is always greater than* v_{rms1} *provided that* $v_{max} > v_{rms1}$ *so that this is a constraint on the rate of increase of* v_{rms}*.*

Exercise 4.7.2. *Suppose that the times and depths to two reflectors are known, say* $\tau_1, \tau_2, z_1,$ *and* z_2*, and that the rms velocities to the reflectors are also known, say* v_{rms1} *and* v_{rms2}*. Consider the interval velocities defined by*

$$v_{int} = \frac{z_2-z_1}{\tau_2-\tau_1} \quad \textit{and} \quad \tilde{v}_{int} = \sqrt{\frac{v_{rms2}^2\tau_2 - v_{rms1}^2\tau_1}{\tau_2-\tau_1}}.$$

Under what condition(s) will these interval velocity estimates be similar? If the interval between the reflectors is highly heterogeneous, which estimate will be larger. Why?

4.8 MATLAB velocity tools

To this point, the discussion of velocity measures has been illustrated with an instantaneous velocity whose form is a known analytic function. More realistically, velocity functions are inherently numerical because they are derived from experiment. The conversion of such numerical velocities

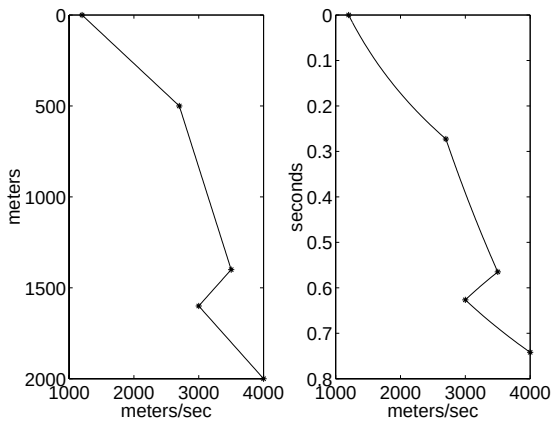

Figure 4.9: (Left) A five legged $v_{ins}(z)$ with linear variation between the knees. (Right) The curve on the left has been converted to $v_{ins}(\tau)$. The variation between the knees is no longer linear.

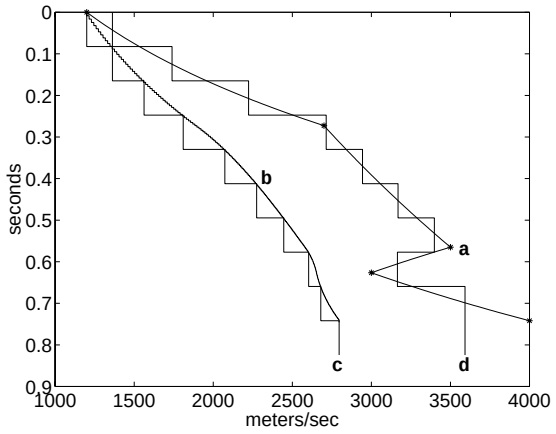

Figure 4.10: (a) $v_{ins}(\tau)$ from Figure 4.9. (b) finely sampled $v_{rms}(\tau)$. (c) de-sampled $v_{rms}(\tau)$. (d) ten legged interval rms approximation to $v_{ins}(\tau)$.

from one form to another is often necessary in data processing and software tools are required for this purpose. Two alternate approaches are (1) fit the numerical velocity functions with an n^{th} order polynomial and perform the necessary integrations using polynomial integration rules, or (2) implement a numerical integration scheme for the conversion formulae. The second approach is taken here.

Five functions are available to convert velocity functions from one form to another. There is also a utility plotting function to draw piecewise constant lines.

vint2t Computes vertical traveltime given interval velocity versus depth.

vint2vave Converts interval velocity to average velocity.

vave2vint Converts average velocity to interval velocity.

vint2vrms Converts interval to rms velocity.

vrms2vint Converts rms velocity to interval velocity.

drawvint Plots interval velocity curves as piecewise constant functions.

No utility is provided to deal with instantaneous velocity since this can be treated simply by generating a finely sampled interval velocity. Like seismic traces, velocity functions are specified by prescribing two vectors, one for the velocities and the other for the depth or time that they apply. Interval velocities are inherently piecewise constant and the convention used here is that the k^{th} velocity, v_k prescribed at depth z_k, persists as a constant until depth z_{k+1}. Therefore, the last velocity in a velocity vector applies for all z greater than the last entry in the depth vector.

As a first example of the use of these functions, suppose that $v_{ins}(z)$ is a piecewise linear function with five "knees" prescribed by (v, z) pairs as (1200 m/s, 0 m), (2700 m/s, 500 m), (3500 m/s, 1400 m), (3000 m/s, 1600 m), (4000 m/s, 2000 m). Code Snippet 4.8.1 shows how to compute $v_{ins}(\tau)$ for

this $v_{ins}(z)$ and displays its results in Figure 4.9. Line 2 uses piecewise linear interpolation (***pwlint***) to resample the five legged function to a fine interval so that the piecewise constant interval velocity approximates the desired piecewise linear instantaneous velocity. Line 5 computes $\tau(z)$ by calling ***vint2t*** and thus defining $v_{ins}(\tau)$. (Note that the ***vint2t*** returns *one-way time* so these values must be doubled for *two-way time*.)

Code Snippet 4.8.1. *This code defines a five legged, piecewise linear $v_{ins}(z)$ and then computes $v_{ins}(\tau)$. It makes Figure 4.9.*

```
1    v=[1200 2700 3500 3000 4000];z=[0 500 1400 1600 2000];
2    z2=0:10:2000;v2=pwlint(z,v,z2);
3    subplot(1,2,1);plot(v2,z2,v,z,'r*');flipy
4    xlabel('meters/sec');ylabel('meters');
5    t2=vint2t(v2,z2);
6    t=interp1(z2,t2,z);
7    subplot(1,2,2);plot(v2,t2,v,t,'r*');flipy;
8    xlabel('meters/sec');ylabel('seconds');
```

End Code

The computation of $\tau(z)$ requires the numerical computation of the integral in equation (4.3). Given a densely sampled function, the simplest way to do a numerical integration is with the functions `sum` and `cumsum`. The difference between these is that the first does a *definite integral* while the second does an *indefinite integral*. For example, the command `sum([1:5].^2)` just adds the squares of the integers from 1 to 5 to get 55. This is a discrete approximation to the definite integral of the function $y = x^2$ from .5 to 5.5 for which the analytic answer is $(5.5^3 - .5^3)/3 \approx 55.4167$. Alternatively `cumsum([1:5].^2)` has the result $[1, 5, 14, 30, 55]$ which gives the value of the integral of $y = x^2$ for a lower limit of .5 and five successive upper limits of 1.5, 2.5, 3.5, 4.5, and 5.5 . Thus `sum` approximates $\int_{.5}^{5.5} x^2 dx$ and is just a single number while `cumsum` approximates $\int_{.5}^{x} \tilde{x}^2 d\tilde{x}$ and results in a vector the same length as the input. Thus, the traveltime integration of equation (4.3) is done within ***vint2t*** with the single command `t1(2:nz)=cumsum(dz./vint(1:nz-1))`. The vector `t1` is $\tau(z)$ and has been initialized to zero, `dz` is a vector of the depth intervals, and `vint` is the interval velocity vector. (For more details, browse the source code file.)

Often velocity functions from well logs do not begin at $z = 0$. Therefore the computation of $\tau(z)$ from $z = 0$ requires that some initial velocity be used for the depth range above the log. This can be input to ***vint2t***, in the form of a constant time shift τ_0, as the fourth argument. τ_0 defaults to $\tau_0 = z(1)/v(1)$, that is the first depth divided by the first velocity. This can be estimated if the depth and time to some marker formation top are already known. Then ***vint2t*** can be run once with the initial velocity set to zero, and the time to the marker can be observed to be off by some $\Delta\tau$ in one-way time.

Now consider the computation of $v_{rms}(\tau)$ and the construction of a ten layer approximation to $v_{ins}(\tau)$ using interval rms velocities. This is done in Code Snippet 4.8.2 and the result is shown in Figure 4.10. Line 2 creates the dense $v_{rms}(\tau)$ function using ***vint2vrms*** shown as curve 'b' in the figure. Line 3 defines a blocky ten legged time vector and line 4 creates the coarse $v_{rms}(\tau)$ (curve 'c') by calling ***vint2vrms*** with a third argument. Essentially, this is just a de-sampled version of the finely sampled $v_{rms}(\tau)$. (It was not necessary to create the finely sampled $v_{rms}(\tau)$ as this was done to demonstrate the software.) Finally line 6 creates the interval rms approximation to $v_{ins}(\tau)$ by sending the coarse $v_{rms}(\tau)$ to ***vrms2vint***.

Code Snippet 4.8.2. *This carries on after Code Snippet 4.8.1 to compute: v_{rms} from the surface for every point in $v_{ins}(\tau)$ a ten point sparse $v_{rms}(\tau)$ and a ten-legged rms-interval velocity approximation to $v_{ins}(\tau)$. Figure 4.10 is the result.*

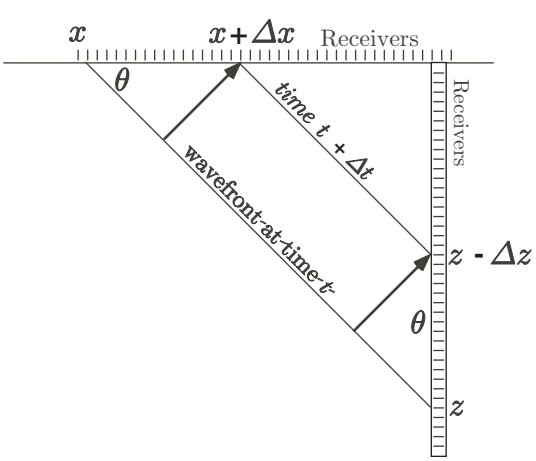

Figure 4.11: An impulsive plane wave approaches horizontal and vertical receiver arrays

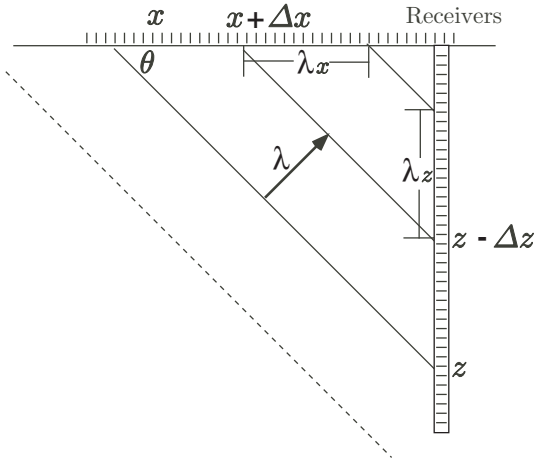

Figure 4.12: Similar to Figure 4.11 except that the plane wave is monochromatic.

```
1    plot(v2,t2,v,t,'r*');flipy
2    vrms=vint2vrms(v2,t2);
3    tblock=linspace(min(t2),max(t2),10);
4    vrmsblock=vint2vrms(v2,t2,tblock);
5    drawvint(t2,vrms);drawvint(tblock,vrmsblock);
6    vint=vrms2vint(vrmsblock,tblock);
7    drawvint(tblock,vint);
8    xlabel('meters/sec');ylabel('seconds');
```

End Code

Exercise 4.8.1. *Load the file* `vp_from_well.mat` *that contains a vector of P-wave velocities and a corresponding vector of depths. Create two ten-leg interval-velocity approximations to* $v_{ins}(z)$ *one using average velocities and one using rms velocities. Compare the accuracy of time to depth conversion using these two alternate approximations. (For time to depth conversions, compute average velocities from both interval velocity functions). Is there any significant difference?*

Exercise 4.8.2. *Create a function called* `vrms2vave` *that converts average velocities to rms velocities. Then create the inverse function* `vave2vrms`*. Test your codes with the universal velocity function by comparing with the analytic forms for* v_{rms} *and* v_{ave}*.*

Exercise 4.8.3. *Working with the universal velocity function, create* $v_{rms}(\tau)$ *for the depth range of 0 to 3000 m. Then use MATLAB's random number generator,* `rand`*, to add a 2% random fluctuation to the* $v_{rms}(\tau)$ *function. Finally calculate* $v_{ins}(\tau)$ *from the perturbed* $v_{rms}(\tau)$ *function and determine the percentage error in the* $v_{ins}(\tau)$ *estimates caused by the 2% error in* $v_{rms}(\tau)$*. What does this say about the sensitivity of interval velocity calculations to noise?*

4.9 Apparent velocity: v_x, v_y, v_z

Consider a wavefront arriving at an array of detectors. For simplicity, use only two dimensions, (x, z), and let the wavefront make an angle θ with respect to the horizontal (θ is called the *emergence angle* of the wave). The detectors are spaced at intervals of Δx at $z = 0$ (Figure 4.11). If the medium

immediately beneath the receivers has the acoustic velocity, v_0, then the wavefront arrives at $x+\Delta x$ later than it does at x by the delay

$$\Delta t = \frac{\sin\theta}{v_0}\Delta x. \tag{4.33}$$

Thus it appears to move along the array at the speed

$$v_x \equiv \frac{\Delta x}{\Delta t} = \frac{v_0}{\sin\theta}. \tag{4.34}$$

The quantity v_x is called an *apparent velocity* because the wavefront appears to move at that speed even though its true speed is v_0. Apparent velocities are one of the four fundamental observables in seismic data, the others being position, time and amplitude. The apparent velocity is never less than the real velocity and can range up to *infinity*. That is $v_0 \le v_x \le \infty$. Since infinities are cumbersome to deal with it is common to work with the inverse of v_x, called the time-dip, $\Delta t/\Delta x$ or horizontal slowness, s_x. Another common convention for horizontal slowness is to call it p which signifies the *ray parameter*.

Now, enlarge the thought experiment to include an array of receivers in a vertical borehole and spaced at Δz. Reasoning similar to that used before shows that the arrival at $z-\Delta z$ is delayed from that at z by

$$\Delta t = \frac{\cos\theta}{v_0}\Delta z. \tag{4.35}$$

Therefore, the vertical apparent velocity, v_z, is

$$v_z \equiv \frac{\Delta z}{\Delta t} = \frac{v_0}{\cos\theta}. \tag{4.36}$$

Using simple trigonometry, it results that

$$\frac{1}{v_0^2} = \frac{1}{v_x^2} + \frac{1}{v_z^2}. \tag{4.37}$$

If this argument is extended to 3D, the result is that there is an apparent velocity in the y direction as well and that the sum of the inverse squares of the apparent velocities is the inverse square of the real velocity:

$$\frac{1}{v_0^2} = \frac{1}{v_x^2} + \frac{1}{v_y^2} + \frac{1}{v_z^2}. \tag{4.38}$$

We observe apparent velocity, or equivalently time-dip, by simply choosing an event of interest on a seismic record and measuring the slope, $\Delta t/\Delta x$. It will be seen shortly that this is a measurement of the ray parameter of the ray associated with the chosen event. Knowledge of the ray parameter essentially determines the raypath, provided that the velocities in the subsurface are known. This allows the ray to be projected down into the earth and to possibly determine it's reflection point. This technique, called *raytrace migration*, will be discussed in section 5.2.2.

Another way to measure apparent velocities is with a multi-dimensional Fourier transform. In 2-D for example, the f-k transform represents a seismic record on a grid of horizontal wavenumber, k_x, and temporal frequency, f. Each point in the (k_x,f) plane has a complex number associated with it that gives the amplitude and phase of a fundamental Fourier component: $e^{2\pi i(k_x x - ft)}$. In 3D these fundamental components are monochromatic (i.e. a single f) plane waves so in 3D or 2D they are called *Fourier plane waves*. These waves are infinite in extent so they have no specific arrival time. However, it does make sense to ask when a particular wave crest arrives at a particular

location. Mathematically, a wave crest is identified as a point of constant phase where *phase* refers to the entire argument of the complex exponential. If the Fourier transform has the value $Ae^{i\phi}$ at some (k_x,f), then the Fourier plane wave at that point is $Ae^{2\pi i(k_x x - ft)+i\phi}$. The motion of a point of constant phase is tracked by equating the phase at (x,t) with that at $(x+\Delta x, t+\Delta t)$ as in

$$2\pi i(k_x x - ft) + i\phi = 2\pi i(k_x(x+\Delta x) - f(t+\Delta t)) + i\phi \tag{4.39}$$

from which it follows that

$$\frac{\Delta x}{\Delta t} = \frac{f}{k_x}. \tag{4.40}$$

Thus the ratio of f to k_x determines the horizontal apparent velocity of the Fourier plane wave at (k_x,f). Radial lines (from the origin) in the (k_x,f) plane connect points of a common apparent velocity. The advantage of the Fourier domain lies in this simultaneous measurement of apparent velocity for the entire seismic section. The disadvantage is that the notion of spatial position of a particular apparent velocity measurement is lost.

If equation (4.38) is evaluated for the Fourier plane wave $e^{2\pi i(k_x x + k_y y + k_z z - ft)}$ the result is

$$\frac{1}{v^2} = \frac{k_x^2}{f^2} + \frac{k_y^2}{f^2} + \frac{k_z^2}{f^2}. \tag{4.41}$$

Now, for any monochromatic wave, we have $\lambda f = v$ and, using $k = \lambda^{-1}$, then equation (4.41) leads to

$$k^2 = k_x^2 + k_y^2 + k_z^2. \tag{4.42}$$

This very important result shows that the coordinate wavenumbers, (k_x,k_y,k_z), are the components of a wavenumber vector, $\vec{k}$. Using a position vector, $\vec{r} = (x,y,z)$, the Fourier plane wave can be written compactly as $e^{2\pi i(\vec{k}\cdot\vec{r} - ft)}$. Equation (4.42) can be expressed in terms of apparent wavelengths (e.g. $k_x = \lambda_x^{-1}, etc.$) as

$$\frac{1}{\lambda^2} = \frac{1}{\lambda_x^2} + \frac{1}{\lambda_y^2} + \frac{1}{\lambda_z^2}. \tag{4.43}$$

which shows that, like apparent velocities, the apparent wavelengths are always greater than the true wavelength. An important property of the wavenumber vector is that it points in the direction of wave propagation (for an isotropic, homogeneous medium). An easy way to see this is to take the gradient of the phase of a Fourier plane wave. Since the wavefront is defined as a surface of constant phase, then wavefronts can be visualized as contours of the phase function $\tilde{\phi} = \pi i(\vec{k}\cdot\vec{r} - ft)$. The gradient of this phase function points in the direction normal to the wavefronts, that is the direction of a raypath. This gradient is

$$\vec{\nabla}\tilde{\phi} = \frac{\partial\tilde{\phi}}{\partial x}\hat{x} + \frac{\partial\tilde{\phi}}{\partial y}\hat{y} + \frac{\partial\tilde{\phi}}{\partial z}\hat{z} \tag{4.44}$$

where $\hat{x}, \hat{y}$, and $\hat{z}$ are unit vectors in the coordinate directions. Calculating the indicated partial derivatives leads to

$$\vec{\nabla}\tilde{\phi} = k_x\hat{x} + k_y\hat{y} + k_z\hat{z} = \vec{k}. \tag{4.45}$$

This equation can be achieved more simply by writing $\vec{\nabla} = \hat{r}\frac{\partial}{\partial r}$, where $r = \sqrt{(}x^2 + y^2 + z^2)$ and $\hat{r}$ is a unit vector pointing to a particular location, so that

$$\vec{\nabla}\tilde{\phi} = \hat{r}\frac{\partial\vec{k}\cdot\vec{r}}{\partial r} = \hat{r}\left|\vec{k}\right| = \vec{k}. \tag{4.46}$$

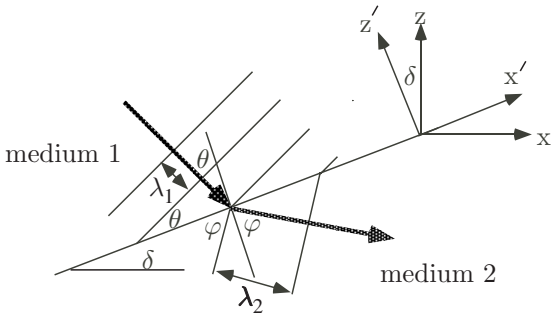

Figure 4.13: Snell's law results from the physical requirement that the apparent velocity along an interface be conserved.

So, the inverse apparent velocities are proportional to the components of the wavenumber vector that points in the direction of wave propagation.

4.10 Snell's Law

Snell's law is the relation that governs the angles of reflection and refraction of wavefronts (or equivalent raypaths) at velocity interfaces. To understand its origins, consider Figure 4.13 that shows a periodic plane wave propagating across a planar interface between two media of velocity v_1 and v_2. In the first medium, the wavelength is $\lambda_1 = v_1/f$ while in the second medium it is $\lambda_2 = v_2/f$. As the wavefronts cross the interface, they must do so in a continuous fashion, otherwise, the wavefronts in the second medium would either lead or lag those in the first medium. Either possibility violates considerations of causality. The only way for the two wavefront systems to maintain continuity across the interface, and still have different wavelengths, is for them to have different angles with respect to the interface. The relationship between these angles follows from a consideration of apparent velocity. Suppose an array of receivers were placed along the interface. Then the requirement of wavefront continuity means that the wavelength component *measured along the interface* must be the same when evaluated in either medium. Since the frequency, f, does not change (this is a property of linear wave propagation), the apparent velocities measured along the interface must be the same in both media. Working in the rotated coordinates (x', z'), the apparent velocity $v_{x'}$ must give the same result when evaluated using v_1 and θ as when using v_2 and ϕ. Thus

$$v_{1x'} \equiv \frac{v_1}{\sin\theta} = v_{2x'} \equiv \frac{v_2}{\sin\phi}. \tag{4.47}$$

This result is called *Snell's law*. The angles involved in Snell's law can be considered as the angles between the wavefronts and the interface or between the raypaths and the normal to the interface.

Snell's law can be derived in other ways, perhaps the most common being to demonstrate that it is a consequence of requiring the raypaths correspond to *stationary traveltimes*. This is known as *Fermat's principle*. Physically, waves can propagate energy from point A to point B along infinitely many different paths. Fermat's principle says that the most important paths are those for which the traveltime is stationary with respect to slight variations of the path. Here, *stationary* most often means a *minimum* though there are important physical situations for which traveltime is maximized.

The derivation of Snell's law given here makes it clear that it applies to reflected as well as

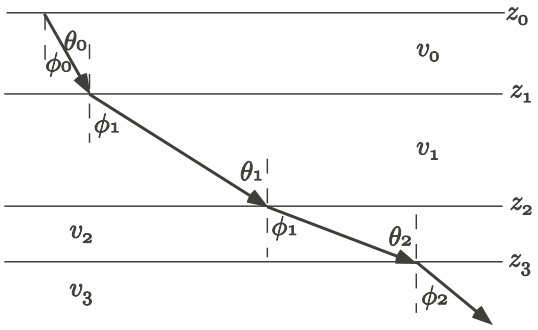

Figure 4.14: In a $v(z)$ medium, all velocity interfaces are horizontal and ray propagation is especially simple.

transmitted energy. In an acoustic medium, where there is only one mode of wave propagation, this results in the angle of incidence and the angle of reflection being equal. However, in elastic media, the incident and reflected waves can be either P-waves or S-waves. For example, in the case of an incident P-wave reflecting as an S-wave, Snell's law states

$$\frac{\sin\theta_p}{v_p} = \frac{\sin\theta_s}{v_s}. \tag{4.48}$$

As a final summary statement, Snell's law states that wavefront propagation across a velocity interface always conserves the apparent velocity along the interface. Though it is not proven here, this holds for arbitrary wave types and for non-planar wavefronts and interfaces.

4.11 Raytracing in a $v(z)$ medium

A $v(z)$ medium is one where $\partial_x v = \partial_y v = 0$ and all velocity interfaces are horizontal. In this case, Snell's law says that the horizontal apparent velocity of the wavefront associated with the ray is conserved. Using the notation of Figure 4.14 this may be stated, for the j$^{\text{th}}$interface, as

$$\frac{\sin\theta_{j-1}}{v_{j-1}} = \frac{\sin\phi_j}{v_j}. \tag{4.49}$$

Since all of the velocity interfaces are horizontal, $\phi_j = \theta_j$ so that

$$\frac{\sin\theta_{j-1}}{v_{j-1}} = \frac{\sin\theta_j}{v_j}. \tag{4.50}$$

This analysis may be repeated at any other interface with a similar conclusion. Thus the quantity $p = \sin\theta_j/v_j$ is conserved throughout the entire wave propagation. p is generally referred to as the *ray parameter* since it is a unique constant for any ray and so *parameterizes* (or names) the possible rays. The conservation of p and its identification as the horizontal apparent slowness is a direct consequence of Snell's law. This analysis generalizes to a continuous variation of v with z such that

$$p \equiv \frac{\sin\left(\theta(z)\right)}{v(z)} = \text{a constant for any particular ray.} \tag{4.51}$$

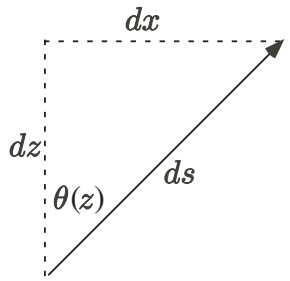

Figure 4.15: A differential ray element ds, at depth z, and travelling at an angle $\theta(z)$ with respect to the vertical.

General expressions for the traveltime and horizontal distance travelled by any ray in a v(z) medium can be easily derived. Figure 4.15 shows a differential element of a ray. From the geometry, it follows that

$$dx = \tan(\theta(z))\, dz \tag{4.52}$$

and

$$dt = \frac{ds}{v(z)} = \frac{dz}{v(z)\cos(\theta(z))}. \tag{4.53}$$

Snell's law is incorporated by replacing the trigonometric functions using $pv(z) = sin(\theta(z))$ so that

$$dx = \frac{pv(z)}{\sqrt{1 - p^2v^2(z)}} dz \tag{4.54}$$

and

$$dt = \frac{dz}{v(z)\sqrt{1 - p^2v^2(z)}}. \tag{4.55}$$

Expressions for macroscopic raypaths are obtained by simply integrating these results. For a ray travelling between depths z_1 and z_2, the horizontal distance travelled becomes

$$x(p) = \int_{z_1}^{z_2} \frac{pv(z)}{\sqrt{1 - p^2v^2(z)}} dz \tag{4.56}$$

and the total traveltime is

$$t(p) = \int_{z_1}^{z_2} \frac{dz}{v(z)\sqrt{1 - p^2v^2(z)}}. \tag{4.57}$$

Given a velocity function and a ray parameter we can compute exactly the horizontal distance (offset) and traveltime for a ray that traverses between two depths. The difficulty is that it is usually desired to trace a ray with a specific offset and there is no simple way to determine the ray parameter that will do this. Generally the process is iterative. The offsets produced by a fan of rays (i.e. p values) are examined and hopefully two p values will be found that bracket the desired offset. Then a new, refined fan of rays can be constructed and the process repeated until a ray is found that produces the desired offset within a specified tolerance (called the capture radius).

If the $v(z)$ medium is discretely layered rather than continuously variable, then summation forms

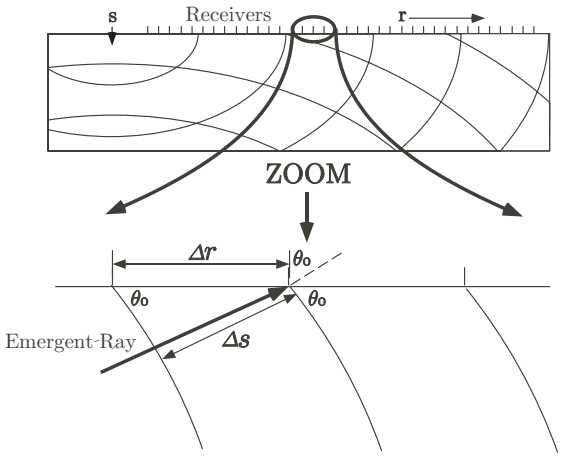

Figure 4.16: The ray parameter can be measured directly by measuring the delay between wavefront arrivals at successive receivers.

of equations (4.56) and (4.57) are more appropriate. These are easily seen to be

$$x(p) = \sum_{k=1}^{n} \frac{pv_k}{\sqrt{1 - p^2 v_k^2}} \Delta z_k \tag{4.58}$$

and

$$t(p) = \sum_{k=1}^{n} \frac{\Delta z_k}{v_k \sqrt{1 - p^2 v_k^2}}. \tag{4.59}$$

4.11.1 Measurement of the ray parameter

Given a seismic source record, the ray parameters for the upcoming waves arriving at the receiver spread may be estimated by measuring the horizontal apparent velocity of each wave (see section 4.9). As shown in Figure 4.16, the emergence angle, θ_0 of a ray arriving at a particular pair of receivers is

$$\sin\theta_0 = \frac{v_0 \Delta t}{\Delta r} \tag{4.60}$$

where v_0 is the instantaneous velocity immediately beneath the receivers, Δr is the receiver spacing, and Δt is the time delay between the wavefront arrival at r and $r + \Delta r$. In arriving at this expression, any wavefront curvature has been assumed to be inconsequential at the local site of measurement. According to equation (4.51), the ray parameter is given by $\sin\theta_0 / v_0$ so

$$\frac{\Delta t}{\Delta r} = \frac{1}{v_r} = \frac{\sin\theta_0}{v_0} = p. \tag{4.61}$$

where v_r is the horizontal apparent velocity of the wave at the measurement location.

Generally, it is expected that p changes with position due to changes in emergence angle and due to lateral variations in the instantaneous velocity. Horizontal events, i.e. waves travelling vertically when they reach the receiver array, have a ray parameter of 0. Waves travelling horizontally across the array have a ray parameter of $1/v_0$ and on a seismic (x, t) plot have the maximum possible

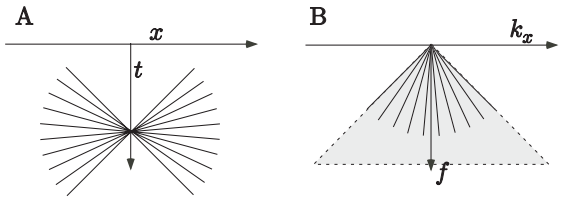

Figure 4.17: The physical considerations that place limits upon ray parameter manifest as limits on the maximum time-dip in (x,t) space (A) and segment (k_x,f) space into allowed and forbidden regions (B).

slope. Even though the wavefronts are vertical, the slope on the time section cannot exceed $1/v_0$. Taking sign into account, the range of possible values for the ray parameter is $-v_0^{-1} \leq p \geq v_0^{-1}$. This maximum possible steepness in (x,t) space is a fundamental property of a seismic time section. Assuming a value for v_0 in the course of data processing means that any events with slopes steeper than v_0^{-1} are interpreted as non-physical and must be rejected. Since apparent velocities can also be expressed in (k_x,f) space as f/k_x (equation (4.40)), this phenomenon means that (k_x,f) space is segmented into allowed and forbidden regions.

Figure 4.17 shows this situation for both (x,t) and (k_x,f) spaces. In (x,t) space, the fan of allowed time dips forms a bow-tie when plotted at a specific point, though such events may exist all (x,t) locations. In (k_x,f) space, the zero ray parameter plots vertically down the f axis while it is horizontal in (x,t). Thus the fan of allowed p values in (k_x,f) forms a symmetric shape about the f axis. (Only one half of the bow tie is shown here because negative frequencies are generally redundant for real-valued data.) In (k_x,f) space, p values are found only along radial lines emanating from the origin, not along lines of the same slope emanating from some other point. The portion of (k_x,f) space corresponding to $|k_x/f| < v_0^{-1}$ (i.e. outside the shaded region in Figure 4.17B) is "forbidden" to simple rays and is called the *evanescent* region. (It will be seen later that certain exponentially decaying wave types can occur here.) The shaded portion of Figure 4.17B is called the *body wave region* and is the portion of (k_x,f) space that is available for seismic imaging.

4.11.2 Raypaths when $v = v_0 + cz$

It is not difficult to integrate equations (4.56) and (4.57) for the case of instantaneous velocity that increases linearly with depth (e.g. the universal velocity function). The details can be found in Slotnick (1959) pages 205-211. Letting $v(z) = v_0 + cz$, $z_1 = 0$, and $z_2 = z$, the results are

$$x(z,p) = \frac{1}{pc}\left[\sqrt{1-p^2v_0^2} + \sqrt{1-p^2\left\{v_0+cz\right\}^2}\right] \tag{4.62}$$

and

$$t(z,p) = \frac{1}{c}\ln\left(\left[\frac{v_0+cz}{v_0}\right]\left[\frac{1+\sqrt{1-p^2v_0^2}}{1+\sqrt{1-p^2\left\{v_0+cz\right\}^2}}\right]\right). \tag{4.63}$$

Slotnick shows that equation (4.62) describes an arc of a circle, having radius $1/(pc)$ and centered at $x_0 = \sqrt{1-p^2v_0^2}/(pc)$ and $z_0 = -v_0/c$. Manifestly, equation (4.62) describes a ray as it goes from zero to some depth z. Since velocity is always increasing in this case, the Snell's law angles are always getting larger as a ray goes deeper. Eventually the ray flattens out, when $\theta(z) = \sin^{-1}(pv(z)) = 90°$,

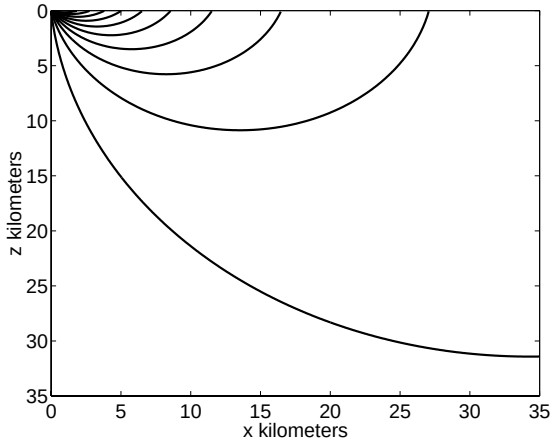

Figure 4.18: A selection of raypaths are shown for the universal velocity function of Figure 4.1. Code Snippet 4.11.1 created this plot.

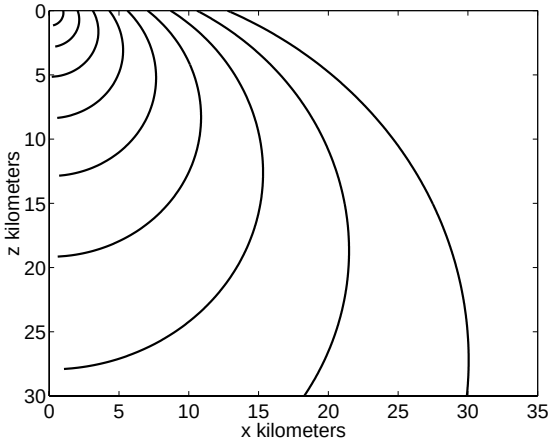

Figure 4.19: A set of wavefronts associated with the raypaths of Figure 4.18. This was created with Code Snippet 4.11.2.

and turns upward. Therefore, the depth at which a ray bottoms out, called its *turning point*, is found from

$$z_{max} = \frac{1 - pv_0}{pc}. \tag{4.64}$$

The complete raypath for a ray that reaches its turning point and then returns to the surface must have two values of x for each z so the function $x(z)$ is mathematically double valued. However, $z(x)$ is still single valued so the complete raypath is most easily computed by solving equation (4.62) for z to get

$$z = \frac{1}{pc}\left[\sqrt{1 - \{pcx - \cos\theta_0\}^2} - pv_0\right]. \tag{4.65}$$

Code Snippet 4.11.1 implements equation (4.65) to create the raypath display shown in Figure 4.18. This code establishes a horizontal distance range and a set of takeoff angles on lines 1-3. Then, in looping over the desired rays, it calculates a depth for each element of the vector `x`. However, many of these rays will not cover the full range of `x`. Equation (4.65) returns a complex number for an `x` distance that cannot be reached by a particular ray. Lines 11-14 search for these points and replace their depths with `NaN`. (Recall that `NaN`'s do not display when plotted.) Also, the code generates z values that are negative so lines 15-18 set these to `NaN`. The ray bending in Figure 4.18 is much more realistic than a simple constant velocity calculation using straight rays. It shows that most of the energy of a seismic source cannot penetrate to great depths because it is turned around by refraction. All of the raypaths are obviously circular arcs whose centers move to the right as the ray parameter decreases.

Code Snippet 4.11.1. *This code implements equation* (4.65) *and makes Figure 4.18.*

```
1    x=1:50:35000;
2    vo=1800;c=.6;nrays=10;
3    thetamin=5;thetamax=80;
4    deltheta=(thetamax-thetamin)/nrays;
5    zraypath=zeros(nrays,length(x));
6    for k=1:nrays
```

```
7           theta=thetamin+(k-1)*deltheta;
8           p=sin(pi*theta/180)/vo;
9           cs=cos(pi*theta/180);
10          z = (sqrt( 1/(p^2*c^2) - (x-cs/(p*c)).^2) -vo/c);
11          ind=find(imag(z)~=0.0);
12          if(~isempty(ind))
13                  z(ind)=nan*ones(size(ind));
14          end
15          ind=find(real(z)<0.);
16          if(~isempty(ind))
17                  z(ind)=nan*ones(size(ind));
18          end
19          zraypath(k,:) = real(z);
20  end
21  figure;plot(x/1000,zraypath/1000);flipy;
22  xlabel('x kilometers');ylabel('z kilometers')
```

End Code

Slotnick also derives expressions for the wavefronts (surfaces of constant traveltime) and shows them to be circles whose centers are along the z axis at

$$z_{w0} = \frac{v_0}{c}\left[\cosh(cT) - 1\right] \tag{4.66}$$

where T is the traveltime defining the wavefront. Each circular wavefront has a radius of

$$r = \frac{v_0}{c}\sinh(cT). \tag{4.67}$$

Code Snippet 4.11.2 calculates and plots 10 wavefronts for a set of traveltimes from 0 to 5 seconds. For each wavefront, the strategy is to calculate the center of the wavefront circle (line z) and its radius (line 8). Then, for a predefined set of depths (line 1) the horizontal positions are calculated from the equation of a circle (line 9). As in the case of the raypaths, this results in both complex and negative values which are found and set to `NaN` (lines 10-17). The resulting wavefronts clearly show the effects of increasing velocity with depth, being more closely spaced at shallow depths than when deeper. Figure 4.20 superimposes the raypaths and wavefronts and uses the command `axis equal` to ensure a 1:1 aspect ratio. Clearly the raypaths are normal to the wavefronts.

Code Snippet 4.11.2. *This code assumes that Code Snippet 4.11.1 has already been run and proceeds from there to calculate and plot the wavefronts. Figure 4.19 is the result.*

```
1   zw=0:50:30000;
2   nwaves=10;tmax=5;
3   xwavefront=zeros(nwaves,length(zw));
4   times=linspace(0,tmax,nwaves);
5   zo=zeros(1,nwaves);
6   for k=1:nwaves
7      zo(k)=vo*(cosh(c*times(k))-1)/c;
8      r=vo*sinh(c*times(k))/c;%radius
9      xw=sqrt(r.^2-(zw-zo(k)).^2);
10     ind=find(real(xw)<0.);
11     if(~isempty(ind))
12        xw(ind)=nan*ones(size(ind));
13     end
14     ind=find(imag(xw)~=0.0);
15     if(~isempty(ind))
16        xw(ind)=nan*ones(size(ind));
```

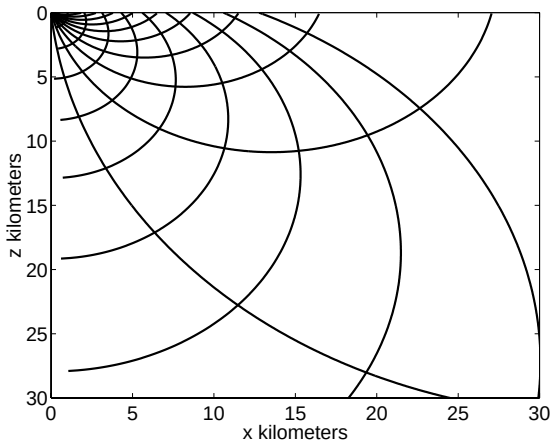

Figure 4.20: This figure superimposes the raypaths of Figure 4.18 onto the wavefronts of Figure 4.19.

```
17      end
18      xwavefront(k,:) = real(xw);
19   end
20   figure;plot(xwavefront/1000,zw/1000);flipy;
21   xlabel('x kilometers');ylabel('z kilometers')
```

End Code

Exercise 4.11.1. *Use equations* (4.62) *and* (4.63) *to calculate and plot the* (x, t) *curve for a P-P reflection from 2000 meters depth. Assume the universal velocity function and a source and receivers at depth 0. Repeat your calculations for a P-S reflection assuming a constant* v_p/v_s *of 2. In each case, what is the maximum source-receiver offset for which a reflection is expected?*

Exercise 4.11.2. *Derive equations* (4.62) *and* (4.63) *from equations* (4.56) *and* (4.57)*.*

Exercise 4.11.3. *For the linear increase of velocity with depth, a source at* $(x, z) = (0, 0)$*, and considering transmitted rays only, is there a unique* p *for each point in the* (x, z) *plane? Will your conclusion remain valid for more general* $v(z)$*?*

4.11.3 MATLAB tools for general $v(z)$ raytracing

The analytic expressions in the previous section produce first-order realism by including the effects of ray bending when $v(z)$ is a linear function of depth. However, more accurate results are often desired for complicated $v(z)$ such as those that result from well log measurements. In this case the only way to proceed is through a numerical implementation of equations (4.56) and (4.57), or more properly, equations (4.58) and (4.59). Usually, it is desired to trace rays between two specific points such as a source and receiver, perhaps via a reflection at a specific depth. There is no known solution technique that solves this *two point raytracing* problem in one step. Instead, an iterative procedure, as alluded to on page 93, must be used.

For definiteness, suppose it is desired to raytrace a P-P reflection from a reflector at $z_r = 2000$m in an arbitrary $v(z)$ medium and for offsets from 100 to 2000 m. Since a P-P ray follows a completely symmetric path (if source and receiver are at the same depth), it is sufficient to trace a ray from

$z = 0$ to $z = z_r$ at offset $x/2$ and then double the results. However, equation (4.56) does not easily facilitate this since the value of p that will do the job is not known. If velocity were constant, then simple geometry would predict the takeoff angle of $\theta_0 = \arctan(x/(2z_r))$ and thus the ray parameter is trivially found. For increasing velocity with depth, the takeoff angle will be smaller than that predicted with constant velocity while for decreasing velocity with depth the situation is reversed. In the general case, the takeoff angle and hence the ray parameter and cannot be found analytically.

There are seven functions and one demonstration script provided for $v(z)$ raytracing. These are

rayfan Shoots a fan of rays given their ray parameters for v(z).

rayfan_a Similar to ***rayfan*** but the rays are specified by angle.

shootray Similar to ***rayfan*** but with less error checking (faster).

traceray_pp Traces a P-P (or S-S) reflection for v(z).

traceray_ps Traces a P-S (or S-P) reflection for v(z).

traceray Traces an arbitrary ray given its raycode for v(z).

drawray Plots rays given their ray parameters.

raytrace_demo Interactive demonstration of v(z) raytracing capabilities.

The approach taken here is to shoot a *fan* of rays and to iteratively refine the fan until a convergence criterion is met. For a general case, this iteration will shoot many fans of rays so the ray fan algorithm needs to be very efficient. The functions ***rayfan***, ***rayfan_a***, and ***shootray*** all performs this task in s similar fashion. However, only the latter is actually used in the two-point raytracers, ***traceray_pp***, ***traceray_ps***, and ***traceray***, because it is the most efficient. However, the efficiency of ***shootray*** is achieved at the sacrifice of some flexibility so ***rayfan*** and ***rayfan_a*** are provided as well.

As shown in Code Snippet 4.11.3, ***shootray*** has three inputs: `v, z`, and `p` that are (respectively) column vectors of velocity and depth and a row vector of ray parameters. The vector of rays defined by `p` is traced from `z(1)` to `z(end)` using the velocities in `v`. As discussed in section 4.8, the velocity model is assumed to be piecewise constant, with velocity `v(k)` beginning at `z(k)` and persisting as a constant until `z(k+1)`. Accordingly, the last velocity in `v` is irrelevant for ***shootray***, and line 3 establishes an index vector to the relevant velocities. Lines 4 through 8 are key to the efficiency of ***shootray***. On line 4, `v(iprop)` is an m length column vector and `p` is an n length row vector. This product of an $m \times 1$ matrix representing $v(z)$ with an $1 \times n$ matrix representing p is an $m \times n$ matrix of $\sin\theta = pv(z)$ called `sn`. The k$^{\text{th}}$ column of `sn` represents the product $pv(z)$ for the k$^{\text{th}}$ ray parameter. Line 6 builds the matrix `cs` that represents $\cos\theta(z) = \sqrt{1 - p^2v^2(z)}$. Lines 7 and 8 build the $m \times n$ matrices `vprop` and `thk` that represent $v(z)$ and Δz for each ray. Since these quantities are the same for each ray, they are represented by matrices with identical columns. Thus, we have four $m \times n$ matrices, with z as the row coordinate and p as the column coordinate. Not all of these combinations of z and p can correspond to physical rays since the depth of penetration of a ray is limited. Line 5 detects these non-physical combinations by finding any values of $pv(z)$ that are greater than one.

Code Snippet 4.11.3. *The function* `shootray` *shoots a fan of rays as defined by the row vector* `p` *from* `z(1)` *to* `z(end)` *through the velocity model defined by the column vectors* `v` *and* `z`.

```
1    function [x,t]=shootray(v,z,p)
2    ... code not displayed ... see shootray.m
3    iprop=1:length(z)-1;
4    sn = v(iprop)*p;
5    [ichk,pchk]=find(sn>1);
6    cs=sqrt(1-sn.*sn);
7    vprop=v(iprop)*ones(1,length(p));
8    thk=abs(diff(z))*ones(1,length(p));
9    if(size(sn,1)>1)
10           x=sum( (thk.*sn)./cs);
11           t=sum(thk./(vprop.*cs));
12   else
13           x=(thk.*sn)./cs;
14           t=thk./(vprop.*cs);
15   end
16   %assign infs
17   if(~isempty(ichk))
18           x(pchk)=inf*ones(size(pchk));
19           t(pchk)=inf*ones(size(pchk));
20   end
```

End Code

The ray tracing is completed in lines 9 through 15 and an `if` statement is used to handle the single layer case. Using the $m \times n$ matrices `thk`, `sn`, `cs`, line 10 computes equation (4.58) by using the `.*` operator to form the expression $pv_k\Delta z_k/\sqrt{1-p^2v_k^2}$ as an $m \times n$ matrix. Then, as discussed in section 4.8, the function `sum` is used to compute the discrete sum that approximates the integral in equation (4.56). When applied to a matrix, `sum` produces a *row* vector that is the sum of each *column* of the matrix. This behavior dictated the construction of the $m \times n$ matrices with p constant in each column. The `if` statement is required for the single layer case also because of the behavior of `sum`. If there is only one layer, then the $m \times n$ matrices are all $1 \times n$ row vectors. When given a row vector, `sum` adds its entries to get a single value instead of "summing" the columns of length 1. The `if` statement circumvents this behavior by omitting the `sum` entirely in the single layer case. The final step, lines 17 through 20 assigns `Inf` to those non-physical values that were flagged earlier on line 5.

The function ***shootray*** does not check for consistency of the inputs to ensure that `v` and `z` are column vectors and `p` is a row vector. Also, it lacks flexibility to specify the start and end depths and always shoots the rays from `z(1)` to `z(end)`. ***rayfan*** traces rays with the same scheme as ***shootray*** but incorporates error checking and allows the start and end depths to specified. This makes ***rayfan*** easier to use but slower. ***shootray*** is intended to be used within a larger raytracing scheme while ***rayfan*** is more easily used from the command line.

The two-point ray tracing functions ***traceray_pp***, ***traceray_ps***, and ***traceray*** all work similarly with an iterative scheme involving multiple calls to ***shootray***. The codes are quite intricate and will not be displayed. ***traceray_pp*** and ***traceray_ps*** are constructed to trace P-P and P-S primary reflections and nothing else. ***traceray*** is more general and can trace an arbitrary ray that has multiple bounces (reflections) and mode conversions. All three codes use similar schemes of building an equivalent layered model that allows the problem to be addressed by shooting rays in one direction only. For example, the P-P reflection problem requires a ray to be traced down through a set of layers to a specified depth and then back up again though nearly the same set of layers. (It will be exactly the same set if the source and receiver depths are the same.) Instead of this two-way problem, a one-way problem is set up by determining the layers for the upgoing leg and placing them beneath the reflector in the order that they are encountered (Figure 4.21). The layers containing the source, receiver, and reflector are adjusted in thickness so that ***shootray*** can

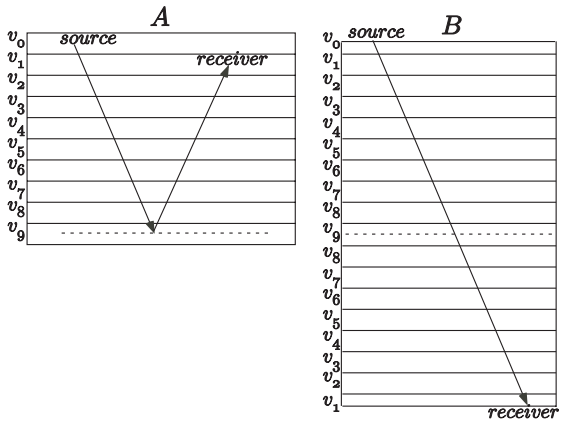

Figure 4.21: (A) A P-P reflection between a source and receiver at different depths. (B) The equivalent one-way raypath to the two-way ray shown in (A). The dashed line is the reflector in both cases.

be used to trace rays from `z(1)` to `z(end)`. A similar equivalent layering can always be constructed for a ray that changes mode (between P and S) and that takes any number of extra bounces. In the former case, the equivalent layering simply must use the correct velocities. In the latter case, the equivalent layering can contain many repeats of a section. This is a technically simple scheme for computing any ray in $v(z)$.

The two-point raytracing functions can all trace rays from a single starting point to multiple receivers at once. That is, they are *vectorized* to produce simple source gathers. Though the source depth need not equal the receiver depth, if multiple receivers are specified, they must all be at the same depth. This means that geometries like that of a VSP must be handled by calling the ray tracer separately for each receiver depth. All three programs iterate until all receivers have a ray traced within a *capture radius* of their position or until a maximum number of iterations is reached. A capture radius is the radius of an imaginary circle around each receiver which, if a ray falls within it, is considered "good enough" for that receiver. If the flag `optflag` is set to 1, then the actual traveltime and ray parameter returned are linearly interpolated between the captured ray and the next closest ray. If `optflag` is zero, then the captured ray is used directly.

Unlike many raytracers, the codes here can reflect, or mode convert, a ray at any depth not just at a layer boundary. Though this may not be physically correct, it allows the common (and very practical) practice of separating the reflectivity from the propagation model. The idea is that the rays are traced through a simplified *background medium* which is chosen to give sufficiently accurate traveltimes for the task at hand. Reflections are assumed to come from a level of detail beyond that required for the traveltime calculations. For example, a linear variation of $v(z)$ may be used with locally determined constants as a background velocity function. Of course, the user who disagrees with this approach is free to prescribe reflections at actual velocity layer boundaries.

Code Snippet 4.11.4 exhibits the use of ***traceray_pp*** and ***traceray_ps*** to model P-P and P-S reflections with the universal velocity function as the background medium. The results are shown in Figures 4.22 and 4.23. Line 1 defines the v_p and v_s velocity models with a v_p/v_s of 2. Lines 2 and 3 establish the basic geometry of source, receivers, and reflector. The capture radius is set to 10% of the receiver spacing and the maximum number of iterations is set to 4 which is adequate for smooth media. The call to ***traceray_pp*** is on line 7 and the final parameter `dflag` instructs the

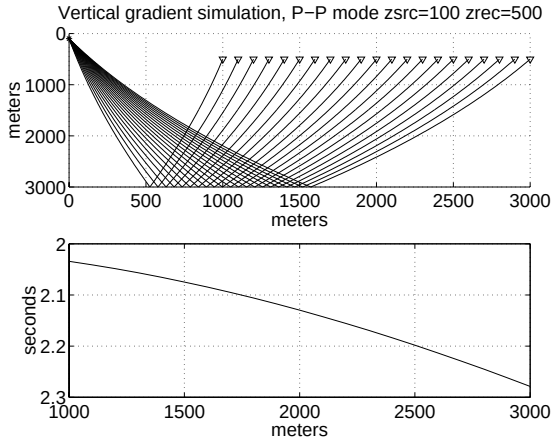

Figure 4.22: A P-P reflection is shown for a background medium of $v_p(z) = 1800 + .6z$m/s. See Code Snippet 4.11.4.

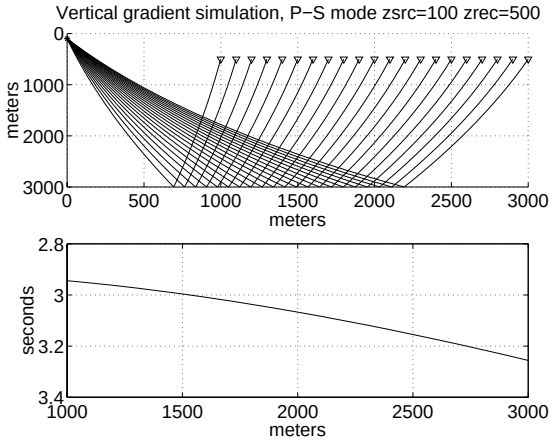

Figure 4.23: A P-S reflection is shown for a background medium of $v_s(z) = 900 + .3z$. See Code Snippet 4.11.4

program to draw the rays into the current figure window. Lines 9 and 10 draw symbols indicating the source and receiver and line 12 plots the traveltime in the bottom half of the figure. Lines 15-22 are very similar except that ***traceray_ps*** is called to create a P-S reflection. An S-P reflection could be created by reversing the order of the velocity arguments (i.e.`traceray_ps(vs,zs,vp,zp,...)`). Similarly, an S-S reflection can be modelled using ***traceray_pp*** and providing it with the S-wave velocity functions.

Code Snippet 4.11.4. *This example raytraces a P-P and a P-S reflection in a linear gradient medium. It creates Figures 4.22 and 4.23.*

```
1    zp=0:10:4000;vp=1800+.6*zp;vs=.5*vp;zs=zp;% velocity model
2    zsrc=100;zrec=500;zd=3000;%source receiver and reflector depths
3    xoff=1000:100:3000;caprad=10;itermax=4;%offsets, cap radius, and max iter
4    pfan=-1;optflag=1;pflag=1;dflag=2;% default ray fan, and various flags
5    % create P-P reflection
6    figure;subplot(2,1,1);flipy;
7    [t,p]=traceray_pp(vp,zp,zsrc,zrec,zd,xoff,caprad,pfan,itermax,optflag,...
8       pflag,dflag);
9    title(['Vertical gradient simulation, P-P mode zsrc=' ...
10      num2str(zsrc) ' zrec=' num2str(zrec)])
11   line(xoff,zrec*ones(size(xoff)),'color','b','linestyle','none','marker','v')
12   line(0,zsrc,'color','r','linestyle','none','marker','*')
13   grid;xlabel('meters');ylabel('meters');
14   subplot(2,1,2);plot(xoff,t);grid;
15   xlabel('meters');ylabel('seconds');flipy
16   % P-S reflection
17   figure;subplot(2,1,1);flipy;
18   [t,p]=traceray_ps(vp,zp,vs,zs,zsrc,zrec,zd,xoff,caprad,pfan,itermax,...
19      optflag,pflag,dflag);
20   title(['Vertical gradient simulation, P-S mode zsrc=' ...
21      num2str(zsrc) ' zrec=' num2str(zrec)])
22   line(xoff,zrec*ones(size(xoff)),'color','b','linestyle','none','marker','v')
23   line(0,zsrc,'color','r','linestyle','none','marker','*')
24   grid;xlabel('meters');ylabel('meters');
```

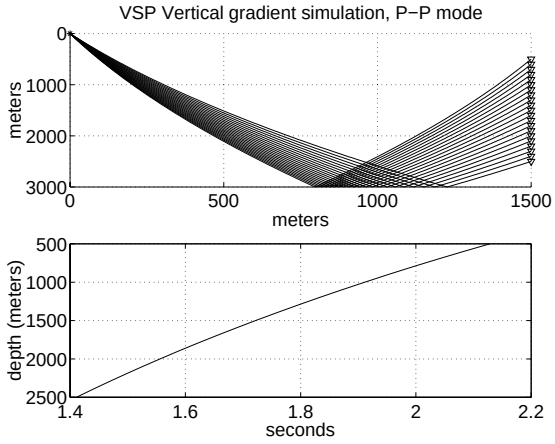

Figure 4.24: A P-P reflection is shown for an offset VSP. See Code Snippet 4.11.5.

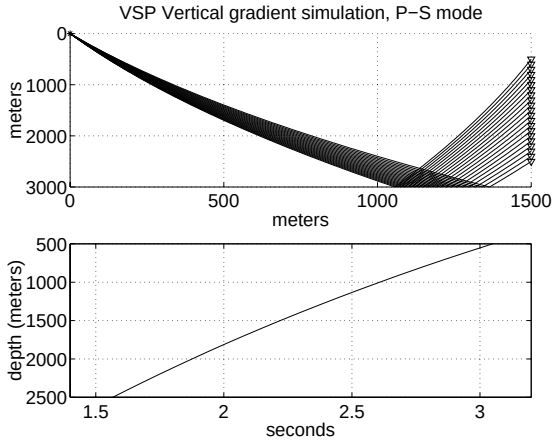

Figure 4.25: A P-S reflection for an offset VSP is shown. See Code Snippet 4.11.5.

```
25    subplot(2,1,2);plot(xoff,t);grid;
26    xlabel('meters');ylabel('seconds');flipy;
```

End Code

Often very interesting calculations can be done by calling these functions in a loop. For example, since the receivers are required to be at a constant depth on each call to ***traceray_pp*** a VSP cannot be modelled with a single call. However, as Code Snippet 4.11.5 shows, the desired results can be obtained by looping over receiver depth. Though not as efficient in this mode, the calculation is still simple and accurate. The parameter `pfan` is set to -2 in this case which causes the ray tracer to start with the final ray fan determined the last time it was called. The modelled events are shown in Figures 4.24 and 4.25.

Code Snippet 4.11.5. *This code traces P-P and P-S reflections for an offset VSP by looping over receiver depth. It creates Figures 4.24 and 4.25.*

```
1     zp=0:10:4000;vp=1800+.6*zp;vs=.5*vp;zs=zp;%velocity model
2     zrec=500:100:2500;zsrc=0;zd=3000;xoff=1500;%geometry
3     caprad=10;itermax=4;%cap radius, and max iter
4     pfan=-2;optflag=1;pflag=1;dflag=2;% default ray fan, and various flags
5     figure;subplot(2,1,1);flipy
6     t=zeros(size(zrec));
7     for kk=1:length(zrec);
8             if(kk==1)dflag=-gcf;else;dflag=2;end
9             [t(kk),p]=traceray_pp(vp,zp,zsrc,zrec(kk),zd,xoff,caprad,pfan,...
10          itermax,optflag,pflag,dflag);
11    end
12    title([' VSP Vertical gradient simulation, P-P mode '])
13    line(xoff,zrec,'color','b','linestyle','none','marker','v')
14    line(0,zsrc,'color','r','linestyle','none','marker','*')
15    grid;xlabel('meters');ylabel('meters');
16    subplot(2,1,2);plot(t,zrec);
17    xlabel('seconds');ylabel('depth (meters)');grid;flipy;
18
19    figure;subplot(2,1,1);flipy;
```

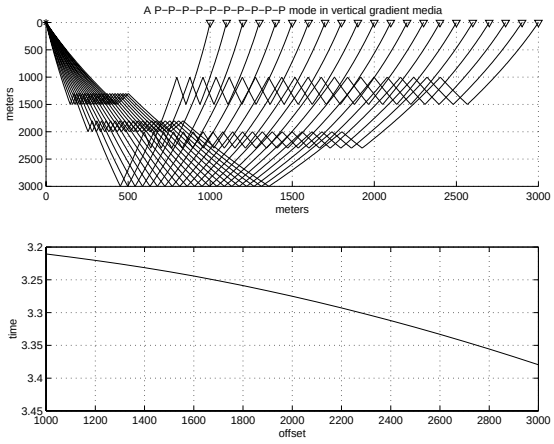

Figure 4.26: A complicated multiple is shown that remains a P-ray throughout. See Code Snippet 4.11.6

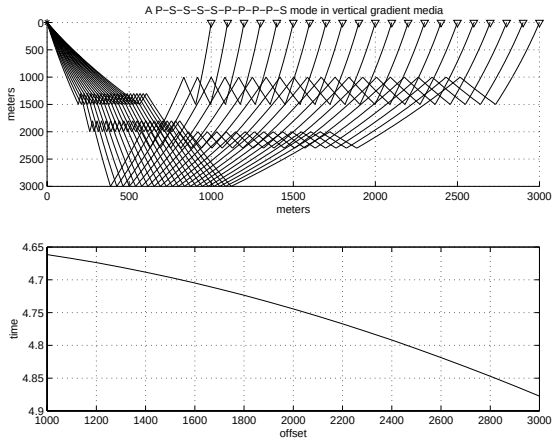

Figure 4.27: A complicated ray is shown that converts back and forth from P to S as it bounces. See Code Snippet 4.11.6

```
20   t=zeros(size(zrec));
21   for kk=1:length(zrec);
22           if(kk==1)dflag=-gcf;else;dflag=2;end
23           [t(kk),p]=traceray_ps(vp,zp,vs,zs,zsrc,zrec(kk),zd,xoff,caprad,pfan,...
24         itermax,optflag,pflag,dflag);
25   end
26   title([' VSP Vertical gradient simulation, P-S mode ']) 
27   line(xoff,zrec,'color','b','linestyle','none','marker','v')
28   line(0,zsrc,'color','r','linestyle','none','marker','*')
29   grid;xlabel('meters');ylabel('meters');
30   subplot(2,1,2);plot(t,zrec);
31   xlabel('seconds');ylabel('depth (meters)');grid;flipy;
```

End Code

As a final example of these raytracing facilities, consider the calculation of a complicated mode that takes multiple bounces and converts back and forth between P and S modes. This calculation can be done with ***traceray*** and is shown in Code Snippet 4.11.6. The key idea here is to use a *ray code* that defines the depths and mode types of the different legs of the raypath. A ray code is an $n \times 2$ matrix with the first column being a list of depths and the second containing either the integer 1 (P-wave) or 2 (S-wave). For example, the ray code `[0 1;1000 2;800 2;1000 1;100 1]` specifies a P-ray from depth 0 to 1000 m. At 1000 m it converts to an S-ray and travels back up to 800 m and then back down to 1000 m while remaining an S-ray. On the second reflection at 1000 m it converts back to a P-ray and travels up to 100 m where it ends. The final entry in column two is meaningless. Code Snippet 4.11.6 demonstrates this facility by creating a complicated multiple bounce P-ray (no mode conversions) on line 8 and a multimode ray on line 17. Generally, it might be expected that significant rays like this will be symmetric (i.e. the same bounce pattern on both the up and down legs); however, creating an asymmetric ray is perhaps a better demonstration.

There are many more possible ways to use these raytracing facilities. For more examples, execute the script ***raytrace_demo*** and follow the on-screen instructions. The code of this script is similar to the examples presented here and should be comprehensible.

Code Snippet 4.11.6. *Here is a demonstration of the use of* `traceray` *to compute multiple-bounce and multiple-mode raypaths. This code creates Figures 4.26 and 4.27.*

```
zp=0:10:4000;vp=1800+.6*zp;vs=.5*vp;zs=zp;
xoff=1000:100:3000;
caprad=10;itermax=4;%cap radius, and max iter
pfan=-1;optflag=1;pflag=1;dflag=2;% default ray fan, and various

raycode=[0 1;1500 1;1300 1;2000 1;1800 1;3000 1;2000 1;2300 1;1000 1; 1500 1; 0 1];
figure;subplot(2,1,1);flipy
[t,p]=traceray(vp,zp,vs,zs,raycode,xoff,caprad,pfan,itermax,optflag,pflag,dflag);
title('A P-P-P-P-P-P-P-P-P-P mode in vertical gradient media');
xlabel('meters');ylabel('meters')
line(xoff,zeros(size(xoff)),'color','b','linestyle','none','marker','v')
line(0,0,'color','r','linestyle','none','marker','*');grid
subplot(2,1,2);plot(xoff,t);
grid;flipy;xlabel('offset');ylabel('time')

raycode=[0 1;1500 2;1300 2;2000 2;1800 2;3000 1;2000 1;2300 1;1000 1; 1500 2; 0 1];
figure;subplot(2,1,1);flipy
[t,p]=traceray(vp,zp,vs,zs,raycode,xoff,caprad,pfan,itermax,optflag,pflag,dflag);
title('A P-S-S-S-S-P-P-P-P-S mode in vertical gradient media');
xlabel('meters');ylabel('meters')
line(xoff,zeros(size(xoff)),'color','b','linestyle','none','marker','v')
line(0,0,'color','r','linestyle','none','marker','*');grid
subplot(2,1,2);plot(xoff,t);
grid;flipy;xlabel('offset');ylabel('time')
```

End Code

Exercise 4.11.4. *Consider the velocity model defined by*

$$v_p(z) = \begin{cases} 1860 + .6z & \dots 0 \le z < 1000m \\ 3100 + .4(z - 1000) & \dots 1000 \le z < 1550m \\ 2750 + .7(z - 1550) & \dots 1550 \le z \end{cases}$$

and v_p/v_s of 2. Create a display showing the P-P and P-S raypaths for source-receiver offsets from 10 to 3010 at 100 meter intervals for a reflector at a depth of 2500 m. Let the source and receivers be a $z = 0$ Make another plot showing the angle of incidence on the reflector (in degrees) versus offset for both P-P and P-S reflections.

Exercise 4.11.5. *For the velocity model in the previous exercise, what is the largest offset that can be successfully raytraced for the reflector at $z = 2500m$ for both P-P and P-S reflections. Explain why the ray tracing fails. How do these conclusions change if the reflector is at 1500 m?*

4.12 Raytracing for inhomogeneous media

Raytracing for more general settings that the v(z) method described in the previous setting can be done by detailing the velocity field on a grid in two or three dimensions and then shooting rays through it. The ray trajectories are found by solving a certain differential equation that can be derived from the wave equation. An alternative to the gridded velocity model is to specify the geometry of geologic interfaces between different velocity units (layers) and then to prescribe the velocity of each unit by a simple analytic function (e.g. constant velocity layers). Rays are then traced through the model by using analytic results within with layers and explicit Snell's law

calculations at the interfaces. The first approach is generally simpler because the raytracing can be reduced to the solution of an ordinary differential equation that implicitly incorporates Snell's law. In the second approach, the description of the geologic interfaces must be done with great care to ensure that they connect without overlap or voids and this can require complex geometric calculations. Furthermore, it is generally a complex matter to determine the next interface that a ray will encounter. However, for three dimensional simulations, the gridded model approach can rapidly consume a great deal of computer memory and can rapidly become impractical. This is especially true if rays are to be traced through a complex structure that lies far beneath a simple one. In this case, the grid must be made quite small for the complex structure which results in too much detail for the upper medium. In this book, only the gridded approach in two dimensions will be explored.

4.12.1 The ray equation

A reasonable expectation for inhomogeneous media is that individual temporal frequencies can be represented with a mathematical form that is similar to that for a Fourier plane wave. In two or three dimensions, a Fourier plane wave has the form $Ae^{2\pi i(ft\pm\vec{k}\cdot\vec{x})}$ where $\vec{k}$ and $\vec{x}$ are the wavenumber and position vectors. By analogy, for the variable-velocity scalar wave equation

$$\nabla^2\psi - \frac{1}{v^2(\vec{x})}\frac{\partial^2\psi}{\partial t^2} = 0 \tag{4.68}$$

an approximate plane-wave solution will now be assumed in the form

$$\psi(\vec{x},t) = A(\vec{x})e^{2\pi if(t-T(\vec{x}))} \tag{4.69}$$

where $A(\vec{x})$ and $T(\vec{x})$ are unknown functions describing amplitude and traveltime that are expected to vary with position. In the constant velocity limit, A becomes constant while T still varies rapidly which leads to the expectation that variation in A will often be negligible compared with variation in T. Substitution of equation (4.69) into equation (4.68) will require computation of the Laplacian of the assumed solution. This is done as

$$\nabla^2\psi(\vec{x},t) = \vec{\nabla}\cdot\vec{\nabla}\left[A(\vec{x})e^{2\pi if(t-T(\vec{x}))}\right] = \vec{\nabla}\cdot\left[e^{2piif(t-T)}\vec{\nabla}A - 2\pi iAe^{2\pi if(t-T)}\vec{\nabla}T\right] \tag{4.70}$$

which can be expanded as

$$\nabla^2\psi(\vec{x},t) = \left\{\nabla^2A - 4\pi if\vec{\nabla}A\cdot\vec{\nabla}T - 4\pi^2f^2A\left[\vec{\nabla}T\right]^2 - 2\pi ifA\nabla^2T\right\}e^{2\pi if(t-T(\vec{x}))}. \tag{4.71}$$

Then, using this result and $\partial_t^2\psi(\vec{x},t) = -4\pi^2f^2Ae^{2\pi if(t-T(\vec{x}))}$ in equation (4.68) and equating real and imaginary parts gives the two equations

$$\left[\vec{\nabla}T\right]^2 - \frac{\nabla^2A}{4\pi^2f^2A} - \frac{1}{v(\vec{x})^2} = 0 \tag{4.72}$$

and

$$\frac{A}{2}\nabla^2T - \vec{\nabla}A\cdot\vec{\nabla}T = 0. \tag{4.73}$$

So far, these results are exact and no simplification has been achieved. However, the second term in equation (4.72) is expected to be negligible when velocity gradients are weak or when frequencies

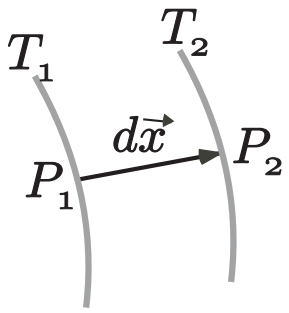

Figure 4.28: A differential raypath element $d\vec{x}$ extends from a point P_1 on a wavefront at time T_1 to a point P_2 on a wavefront at time T_2. The ray segment is perpendicular to both wavefronts and has length $|d\vec{x}| = \Delta s$.

are high regardless of the velocity gradient. When this term is discarded, the result is the nonlinear, partial-differential equation called the *eikonal equation*

$$\left[\vec{\nabla}T\right]^2 - \frac{1}{v(\vec{x})^2} = 0. \tag{4.74}$$

Though not exact, for many realistic situations, a solution to the eikonal equation gives accurate traveltimes through complex media. Equation (4.73) is called the *geometrical spreading equation* because its solution can be shown to describe the flow on energy along a raypath.

The differential equation for raypaths is the vector equation that is implied by the eikonal equation (4.74). For isotropic media, raypaths are normal to the wavefronts and the latter are described as surfaces where $T(\vec{x}) = constant$. Therefore, $\vec{\nabla}T$ must be a vector that points in the direction of the raypath and the eikonal equation shows that $\left|\vec{\nabla}T\right| = v^{-1}$. Consider a wavefront at time $T_1(\vec{x}_1)$ and another at a slightly later time $T_2(\vec{x}_2)$ where $T_2 - T_1$ is very small. Let P_1 be a point on the surface T_1 and P_2 be the corresponding point on T_2 along the normal from P_1 (Figure 4.28). Then $\Delta s = (T_2(P_2) - T_1(P_1))/v(P_1)$ is an estimate of the perpendicular distance between these wavefronts, and

$$\vec{\nabla}T(P_1) = \lim_{P_2 \to P_1} \frac{T_2(P_2) - T_1(P_1)}{\Delta s}\hat{s} = \frac{\hat{s}}{v(P_1)} \tag{4.75}$$

where $\hat{s}$ is a unit vector that is normal to the surface $T_1(P_1)$ or, in other words, $\hat{s}$ points along the raypath. If $d\vec{x}$ is the differential vector pointing from P_1 to P_2 then $\hat{s}$ may be written

$$\hat{s} = \frac{d\vec{x}}{ds} \tag{4.76}$$

where $ds = |d\vec{x}|$ and s is the arclength along the raypath. Combining equations (4.75) and (4.76) gives the *raypath equation*

$$\vec{\nabla}T(\vec{x}) = \frac{\hat{s}}{v(\vec{x})} = \frac{1}{v(\vec{x})}\frac{\mathrm{d}\vec{x}}{\mathrm{d}s}. \tag{4.77}$$

The traveltime T can be eliminated from equation (4.77) by calculating its derivative with respect to arclength, which is

$$\frac{\mathrm{d}\vec{\nabla}T}{\mathrm{d}s} = \frac{\mathrm{d}}{\mathrm{d}s}\frac{1}{v}\frac{\mathrm{d}\vec{x}}{\mathrm{d}s}. \tag{4.78}$$

The left-hand-side of this equation can be simplified as follows

$$\frac{\mathrm{d}\vec{\nabla}T}{\mathrm{d}s} = \vec{\nabla}\left[\frac{\mathrm{d}T}{\mathrm{d}s}\right] = \vec{\nabla}\left[\frac{1}{v(\vec{x})}\right]. \tag{4.79}$$

The last step in this equation is justified using equation (4.77) and the identity $d/ds = \hat{s}\cdot\vec{\nabla}$. Intuitively, this step is valid because $\vec{\nabla}T$ points along the raypath and therefore dT/ds, that is the scalar derivative with respect to arclength along the raypath, gives the full magnitude of $\vec{\nabla}T$. Thus the ray equation is recast as

$$\frac{1}{v^2(\vec{x})}\vec{\nabla}v(\vec{x}) = -\frac{\mathrm{d}}{\mathrm{d}s}\frac{1}{v(\vec{x})}\frac{\mathrm{d}\vec{x}}{\mathrm{d}s} \tag{4.80}$$

which is a second order ordinary differential equation for the raypath vector $\vec{x}$. This result can be further recast into a system of first-order equations by modifying the right-hand-side using

$$\frac{\mathrm{d}}{\mathrm{d}s} = \frac{\mathrm{d}t}{\mathrm{d}s}\frac{\mathrm{d}}{\mathrm{d}t} = \frac{1}{v(\vec{x})}\frac{\mathrm{d}}{\mathrm{d}t} \tag{4.81}$$

and defining the slowness vector, $\vec{p} = \vec{\nabla}T$, that is (from equation (4.77)

$$\vec{p} = \frac{1}{v(\vec{x})}\frac{\mathrm{d}\vec{x}}{\mathrm{d}s} = \frac{1}{v^2(\vec{x})}\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}. \tag{4.82}$$

These last two equations allow equation (4.80) to be recast as the first-order system

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = v^2(\vec{x})\vec{p} \tag{4.83}$$

and

$$\frac{\mathrm{d}\vec{p}}{\mathrm{d}t} = -\frac{\vec{\nabla}v(\vec{x})}{v(\vec{x})}. \tag{4.84}$$

Verification that these two equations (4.83) and (4.84) are equivalent to equation (4.80) can be done by solving equation (4.83) for $\vec{p}$ and substituting it into equation (4.84).

Example 4.12.1. *As an illustration of the validity of equations* (4.83) *and* (4.84)*, it is instructive to show that they reduce to the $v(z)$ case considered previously. If $v(\vec{x}) = v(z)$, then in two dimensions, let $\vec{x} = [x, z]$ and $\vec{p} = [p_x, p_z]$ so that equation* (4.84) *becomes*

$$\frac{\mathrm{d}p_x}{\mathrm{d}t} = 0 \quad \textit{and} \quad \frac{\mathrm{d}p_z}{\mathrm{d}t} = -\frac{1}{v(z)}\frac{\partial v(z)}{\partial z} \tag{4.85}$$

and equation (4.83) *is*

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v^2(z)p_x \quad \textit{and} \quad \frac{\mathrm{d}z}{\mathrm{d}t} = v^2(z)p_z. \tag{4.86}$$

The first of equations (4.85) *immediately integrates to $p_x = constant$. Using this result in the first of equations* (4.86) *together with $v^{-2}(z)\mathrm{d}x/\mathrm{d}t = v^{-1}(z)\mathrm{d}x/\mathrm{d}s$ results in*

$$p_x = \frac{1}{v(z)}\frac{\mathrm{d}x}{\mathrm{d}s} = \frac{\sin\theta}{v(z)} = constant \tag{4.87}$$

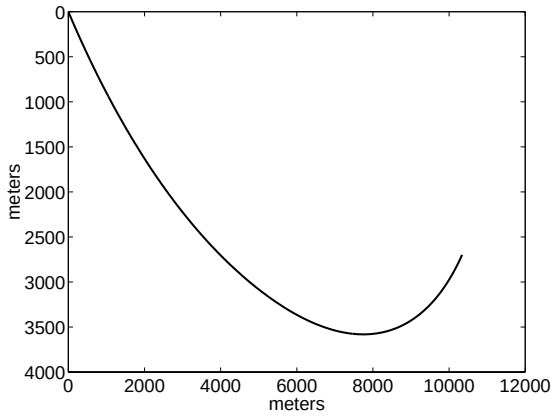

Figure 4.29: A ray is shown traced through $v(x,z) = 1800 + .6z + .4x$ using `ode45` as shown in Code Snippet 4.12.1.

where $\sin\theta = \mathrm{d}x/\mathrm{d}s$ *has been used. Of course, this is Snell's law for the* $v(z)$ *medium. Of the remaining two equations in the system* (4.85) *and* (4.86)*, the first is redundant because* $p_x^2 + p_z^2 = v^{-2}(z)$ *and the second gives*

$$\mathrm{d}t = \frac{\mathrm{d}z}{v^2(z)p_z} = \frac{\mathrm{d}z}{v(z)\cos\theta} = \frac{\mathrm{d}z}{v(z)\sqrt{1 - v^2(z)p_x^2}}. \tag{4.88}$$

When integrated, this will result in equation (4.57)*. In that equation,* p *is the same as* p_x *here.*

4.12.2 A MATLAB raytracer for $v(x,z)$

The numerical solution of equations (4.83) and (4.84) can be done using well-tested and very general methods for solving first-order ordinary differential equations. An example is the fourth-order Runge-Kutta method (Press et al. (1992) chapter 16) that will be used here. This and other general methods are available in MATLAB; however, it is often useful to implement a specific Runge-Kutta scheme to gain improved flexibility.

To proceed, define the abstract *ray vector* $\vec{r} = [\vec{x},\ \vec{p}]$. That is, in n dimensions $\vec{r}$ is a $2n$ dimensional vector formed by concatenating the position and slowness vectors. In two dimensions, the ray vector is $\vec{r} = [x\ z\ p_x\ p_z]$ and the time derivative of $\vec{r}$ is defined through equations (4.83) and (4.84) as

$$\frac{\mathrm{d}r(1)}{\mathrm{d}t} = v^2 r(3); \quad \frac{\mathrm{d}r(2)}{\mathrm{d}t} = v^2 r(4); \quad \frac{\mathrm{d}r(3)}{\mathrm{d}t} = -\frac{\partial \ln v}{\partial x}; \quad \frac{\mathrm{d}r(4)}{\mathrm{d}t} = -\frac{\partial \ln v}{\partial z}. \tag{4.89}$$

Defining the vector $\vec{a} = [v^2\vec{p}\ \ -\vec{\nabla}(\ln v)]$, equation (4.89) becomes

$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}t} = \vec{a}. \tag{4.90}$$

Then, given this prescription for $\mathrm{d}\vec{r}/\mathrm{d}t$ and initial values for $\vec{r}$, the fourth-order Runge-Kutta (RK4) method is invoked to integrate equation (4.90) for $\vec{r}(t)$.

The MATLAB implementation requires a velocity matrix giving $v(x,z)$ on a grid with $\Delta x =$

$\Delta z \equiv \Delta g$ and uses the convention that $(x = 0, z = 0)$ is in the upper left corner. Prior to raytracing, the function ***rayvelmod*** is invoked with arguments being the velocity matrix and the grid spacing Δg. This function creates a number of global variables that will be used repeatedly in the raytracing. These include matrices of v^2, $\partial_x \ln v$, and $\partial_z \ln v$ that are needed in equation (4.89). This precomputation speeds the raytracing and is especially beneficial if a great many rays are to be traced; however, the cost is three times the memory overhead of the simple velocity matrix. ***rayvelmod*** need only be called once at the beginning of the raytracing unless the velocity model is changed.

The function ***drayvec*** implements the computation of $\mathrm{d}\vec{r}/\mathrm{d}t$ according to equation (4.89). This function is designed with the interface required by MATLAB's built-in ODE solver `ode45`. This latter function implements an RK4 scheme by calling a user-designed function like ***drayvec*** that computes the time derivative of the solution vector. The general interface required by `ode45` for such a function is that it must have two input arguments that are the current time and the current value of the solution vector $\vec{r}$. Thus, even though equation (4.90) does not require the value of t to compute $\mathrm{d}\vec{r}/\mathrm{d}t$, ***drayvec*** requires t as input but does not use it.

Code Snippet 4.12.1. *This code builds a velocity matrix representing $v(x, z) = 1800 + .6z + .4x$ and then uses* ***ode45*** *to trace a ray. It creates Figure 4.29.*

```
1    dg=10; %grid spacing
2    tstep=0:.004:3; %time step vector
3    x=0:dg:10000;z=x'; %x and z coordinates
4    v=1800+.6*(z*ones(1,length(x)))+.4*(ones(length(z),1)*x);%velocity
5    rayvelmod(v,dg);clear v;%initialize the velocity model
6    theta=pi*45/180;%takeoff angle
7    r0=[0,0,sin(theta)/1800,cos(theta)/1800]';%initial value of r
8    [t,r]=ode45('drayvec',tstep,r0);%solve for raypath
9    plot(r(:,1),r(:,2));flipy%plot
```

End Code

Code Snippet 4.12.1 illustrates the tracing of a single ray in a medium with both vertical and horizontal velocity gradients. The time-step vector is established on line 2 and a .004 second timestep is used. Smaller timesteps will improve accuracy but will also lengthen the computation. For a particular velocity model, some testing may be required to determine the optimal time step for the problem at hand. The velocity matrix is built on line 4 and then passed to ***rayvelmod*** on line 5. It is then cleared to save space because ***rayvelmod*** has established the required velocity information in global matrices. The takeoff angle and initial values for $\vec{r}$ are calculated on lines 7 and 8. As mentioned in exercise 4.12.1, the components of $\vec{p}$ are not independent of one another since $\vec{p}\cdot\vec{p} = v^{-2}$ and this is illustrated in the calculation of the initial values. Finally, `ode45` is invoked to integrate equation (4.90) and thus compute $\vec{r}$ for the vector of times established on line 2. The calculated ray is shown in Figure 4.29.

In the computation of $\mathrm{d}\vec{r}/\mathrm{d}t$ it is generally true that the ray coordinates at a particular time will not coincide with grid points in the model. Thus the estimation of the velocity terms in equations (4.89) requires some form of interpolation. By default, ***drayvec*** uses nearest neighbor interpolation because this is the fastest method. For more accuracy, bilinear interpolation is also available and its use is controlled through a global variable that is explained in the ***drayvec*** help file.

A problem with the use of `ode45`, that is apparent upon close inspection of Figure 4.29, is that the ray has been traced beyond the bounds of the velocity model. To avoid indexing into the velocity array beyond its bounds, ***drayvec*** has been constructed to use the first (or last) column to represent a simple $v(z)$ medium for rays than go beyond the beginning (or end) of the model. A similar strategy is used to cope with rays that go beyond the minimum or maximum depths. Though this allows

the solution to proceed, a better approach would be to detect when a ray has reached the boundary of the model and stop the raytracing. For this purpose, an RK4 solver has been built as described in Press et al. (1992) and is incorporated in the functions *shootrayvxz* and *shootrayvxz_g*. In these functions, the ray is tested at each time step to determine if it is within the model bounds and raytracing is halted before the maximum time if the ray leaves the model. The syntax to trace a ray with either program is essentially the same as with Code Snippet 4.12.1 except that the command `[t,r]=shootrayvxz(tstep,r0)` replaces `[t,r]=ode45('drayvec',tstep,r0)` on line 8. The functions *shootrayvxz* and *shootrayvxz_g* differ in that the latter calls *drayvec* to compute $\mathrm{d}\vec{r}/\mathrm{d}t$ while the former does this computation directly without the function call. The result is that *shootrayvxz* is much more efficient but does not offer bilinear interpolation as does *shootrayvxz_g*. If nearest-neighbor interpolation is satisfactory, then *shootrayvxz* should be preferred because it is much faster.

Another useful raytracer is *shootraytosurf*. This function works similarly to *shootrayvxz* but the ray is traced until it reaches $z = 0$ or a maximum time limit is exceeded. This is useful in normal incidence raytrace modelling that will be discussed in chapter 5 in connection with normal incidence raytrace migration.

Chapter 5

Elementary Migration Methods

In exploration seismology, *migration* refers to a multi-channel processing step that attempts to spatially re-position events and improve focusing. The term is essentially synonymous with *imaging* (though the latter term has a specific secondary meaning as a sub-step in a migration process). Before migration, seismic data is usually displayed with traces plotted at the surface location of the receivers and with a vertical time axis. This means that dipping reflections are systematically mispositioned in the lateral coordinate and the vertical time axis needs a transformation to depth. Also problematic is the unfocussed nature of seismic data before migration. It is very much like looking through an unfocussed camera lens. Just as the construction of a camera lens requires knowledge of the physics of light propagation, migration algorithms incorporate the physics of acoustic and elastic wave propagation. In addition to spatial positioning and focusing, migration algorithms perform amplitude and phase adjustments that are intended to correct for the effects of the spreading (or convergence) of raypaths as wave propagate.

Migration can be viewed as an approximate solution to the general elastic wavefield inversion problem (Gray, 1997). Full elastic inversion uses the entire seismic wavefield as input into a mathematical process that seeks to estimate the elastic parameters of the earth. This is called an inverse problem because it is opposite to the classical *forward modelling* problem of predicting the seismic wavefield response of a known elastic earth model. Generally, a forward problem can be reduced to finding the solution to a partial differential equation, in this case the elastic wave equation, given specifications of the coefficients of the equation as functions of position and given appropriate boundary conditions. Though often this is a very difficult process, it is usually more easily accomplished than the corresponding inverse problem. The inverse problem usually amounts to estimating the coefficients of a partial differential equation given its response to a known source. Often, these inverse problems are *ill-posed* which is a mathematical term for a problem whose inputs are insufficient to determine all of its expected outputs. For example, the solution to the constant-velocity migration problem (section 5.4.1) requires two surface measurements, the wavefield (ψ) and its vertical derivative ($\partial_z \psi$), all along the surface. Despite this mathematical requirement, there is no feasible technology for measuring $\partial_z \psi$ so methods must be found to deal with only ψ. In addition to being ill-posed, inversion problems are generally nonlinear and practical schemes are typically linearized approximations. This means that they are very sensitive to an assumed initial model. This is unfortunate because knowledge of the subsurface is very limited and the construction of initial models is extremely difficult.

Thus, for many reasons, migration amounts to a very approximate solution to a general, nonlinear, inverse problem. The need to find approximate solutions has lead to a large number of different

migration algorithms that are generally distinguished by the kind of approximations made. As a result there are many different, overlapping ways to categorize migration algorithms. For example, it can be shown that an assumption that waves are traveling only upward at the earth's surface, though physically incorrect, allows the solution to proceed without knowledge of $\partial_z \psi$. (Or equivalently, this assumption allows $\partial_z \psi$ to be calculated from ψ.) This approach is called the *one-way wave* assumption and is very common. Given that one-way waves will be considered, there are: finite difference algorithms that offer great flexibility with spatial variations of velocity but suffer from *grid dispersion* and *angle limitations*, Kirchhoff methods that can easily accommodate arbitrary variations in seismic recording geometry but require raytracing to guide them, and Fourier (or spectral) techniques that offer high fidelity but have difficulty coping with rapid variations in either velocity or recording geometry. All of these techniques attempt, in some manner, to *downward continue* measurements made at $z = 0$ into the subsurface. There are also useful distinctions about whether the estimation of $R(x, y, z)$ is made directly from $z = 0$, *direct* methods, or whether $R(x, y, z)$ is estimated from $z - \Delta z$, *recursive* methods. Finally, perhaps the most common categorization comes under the confusing labels of *time migration* and *depth migration*. The former is an earlier technology that remains viable because it is very insensitive to velocity errors even though it is known to be accurate only if $\partial_x v \sim 0$. On the other hand, depth migration is the only currently available imaging technology that is accurate when $\partial_x v$ varies strongly; however, it requires a very accurate specification of the velocity model.

These last remarks hint at the general chicken-and-egg nature of modern migration methods. A true inversion technique would derive the velocity model from the data as part of the inversion. Migration requires the velocity model as input and merely attempts to reposition, focus, and adjust amplitudes. These really all turn out to be the same thing. In order to be a useful process, it should be true that migration requires only an approximate, or background, velocity model and that more detailed velocity information can be extracted from a successful migration than was input to it. This is generally the case though it can be very difficult to achieve in structurally complex areas. Especially for depth migration, the construction of the velocity model is the central difficulty in achieving a successful result.

This chapter will discuss "elementary" migration methods. This refers to techniques designed to migrate stacked seismic data (post-stack migration) in simple velocity variations. More complex topics including pre-stack migration will be discussed in the next chapter.

5.1 Stacked data

5.1.1 Bandlimited reflectivity

The ultimate goal of a migration is to transform the seismic data into *bandlimited reflectivity*. In the simplest context, reflectivity means the *normal-incidence reflection coefficient* of P-waves. Potentially, every point in the subsurface has a reflectivity value associated with it. In a one dimensional layered medium, P-wave reflectivity is given by $r_k = 2(I_k - I_{k-1})/(I_k + I_{k-1})$ where $I_k = \rho_k v_k$ is the P-wave impedance of the k^{th} layer. In the continuous case, this can be written as $r_{\delta z}(z) = .5\partial_z \ln(I(z))\delta z$ where δz is a small increment of depth. In a 3D context, reflectivity of a small earth element, δvol, is conveniently described by

$$r(x, y, z) = .5 \left| \vec{\nabla}(\log(I(x, y, z))) \right| \delta vol \tag{5.1}$$

At best, equation (5.1) can only be an expression for normal incidence reflectivity. More generally, reflectivity must be acknowledged to be a function of the angle of incidence as well as spatial position.

For the case of plane elastic waves in layered media, the Zoeppritz equations (Aki and Richards, 1980) are an exact prescription of this variation. The Zoeppritz equations are highly complex and nonlinear and the estimation of their approximate parameters from seismic data is called the study of *amplitude variation with offset* or AVO. (A nearly synonymous term is *amplitude variation with angle* or AVA.) Such complex reflectivity estimates are key to true lithology estimation and are a subject of much current research. Ideally, AVO analysis should be conducted simultaneously with migration or after migration. For the current discussion, it is assumed that only the normal-incidence reflectivity, $r(x, y, z)$, is of interest and that a stacked seismic section provides a bandlimited estimate of $r(x, y, z)$.

The estimate of reflectivity must always be bandlimited to some *signal band* of temporal frequencies. This is the frequency range over which signal dominates over noise. Unless some sort of nonlinear or model-based inversion is done, this signal band is determined by the power spectrum of the source, the data fold, the types of coherent and random noise, and the degree of anelastic attenuation (Q loss). The optimal stacked section will have a zero-phase, white, embedded wavelet. This means that it should obey the simple convolutional model

$$s(t) = r(t) \bullet w(t) \tag{5.2}$$

where w(t) is a zero phase wavelet whose amplitude spectrum is white over some limited passband. If w(t) has residual phase or spectral color then these will adversely affect the resolution of the final migrated image. They should be dealt with before migration.

5.1.2 The zero offset section

A discussion of post-stack migration greatly benefits from a firm theoretical model of the CMP (common midpoint) stack. A simple model of stacked seismic data is the *zero-offset section* or "ZOS" model. This model asserts that the pre-stack processing and CMP stack estimates a signal-enhanced version of what we would have recorded had there been a single, coincident soure/receiver pair at each CMP. Each stacked trace represents a separate physical experiment that can be approximately described by the scalar wave equation (assuming acoustic energy only). Taken together, the ensemble of stacked traces has a more complicated basis in physical theory.

The major steps the in the estimation of the signal enhanced ZOS are

Spherical spreading correction: A time dependent amplitude scaling that approximately corrects for spherical divergence.

Deconvolution: The source signature and the average Q effect is estimated and removed.

Sort to (x, h) coordinates: The data is considered to be gathered in (s, g) (source,geophone) coordinates and then sorted to midpoint, x, and half-offset, h (Figure 5.1).

Statics correction: Near surface static time delays are estimated and removed.

NMO removal: The data is taken through stacking velocity analysis and the resulting velocities are used to remove normal moveout time delays.

Residual statics correction: Residual time delays (small) are sought and removed. Usually this is a surface-consistent step.

Trace balancing: This might be considered optional but some step is needed to correct for source strength and geophone coupling variations.

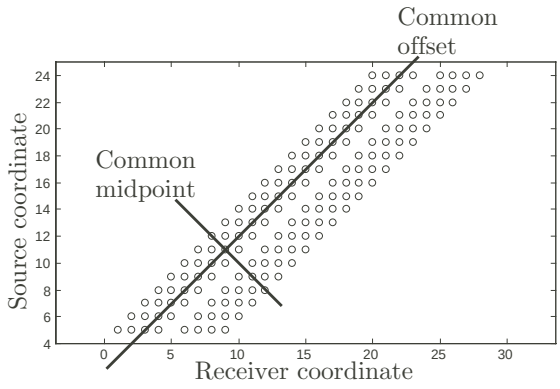

Figure 5.1: A 2D seismic line is depicted in the (s, g) plane. Each shot is represented as having eight receivers with a central gap. Common midpoint, x, and common offset, h coordinates are depicted.

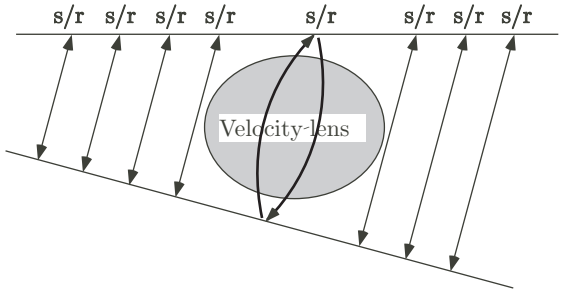

Figure 5.2: Most, but not all, zero-offset raypaths are also normal-incidence raypaths.

CMP stacking: All traces with a common midpoint coordinate are summed. Events which have been flattened on CMP gathers are enhanced. Other events and random noise are reduced.

The ZOS is said to be signal enhanced because the stacking process has been designed to select against multiples. The simplest (and dominant) raypath which returns energy to its source is called the *normal-incidence* raypath. Quite obviously, the normal-incidence reflection will send energy back up along the path that it traveled down on. However, it is possible to have zero-offset paths that are not normal-incidence paths (Figure 5.2).

5.1.3 The spectral content of the stack

The sort from (s, g) to (x, h) marks the transition from single channel to surface consistent processing. Where it actually takes place is somewhat arbitrary because a seismic processing system can always resort the data whenever required. However, it is a logical transition that must occur somewhere before stack. (x, h) coordinates are defined by

$$x = \frac{s+g}{2} \quad \text{and} \quad h = \frac{g-s}{2} \tag{5.3}$$

of which the inverse transformation is

$$s = x - h \quad \text{and} \quad g = x + h. \tag{5.4}$$

The relationship between (s, g) and (x, h) is depicted graphically in Figure 5.1.

Mathematically, a 2D pre-stack seismic dataset is a 3D function, $\psi(s, g, t)$. This dataset can be written as an inverse Fourier transform of its spectrum through

$$\psi(s, g, t) = \int_{\mathbf{V}_\infty} \phi(k_s, k_g, f) e^{2\pi i(k_s s + k_g g - ft)} dk_s dk_g df \tag{5.5}$$

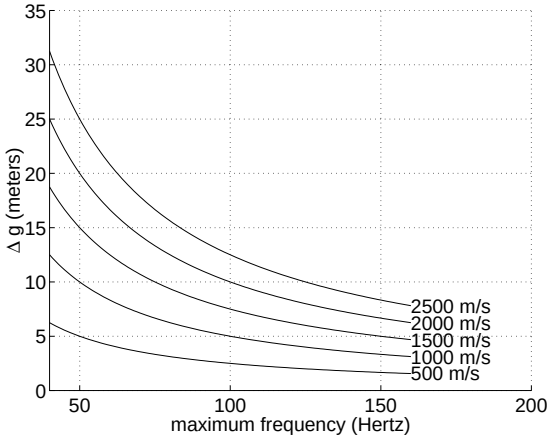

Figure 5.3: These curves show the maximum spatial sample rate required to avoid aliasing of k_g on shot records (assuming point receivers) as a function of the maximum signal frequency. A curve is drawn for each of five near surface velocities. This is based on expression (5.15).

where the notation V_∞ indicates the integration covers the entire relevant, possibly infinite, portion of (k_s, k_g, f) space. Imposing the coordinate transform of equations 5.4 gives

$$\psi(x,h,t) = \int\limits_{\mathbf{V}_\infty} \phi(k_s,k_g,f) e^{2\pi i(k_s(x-h)+k_g(x+h)-ft)} dk_s dk_g df \tag{5.6}$$

or

$$\psi(x,h,t) = \int\limits_{\mathbf{V}_\infty} \phi(k_s,k_g,f) e^{2\pi i((k_s+k_g)x+(k_s-k_g)h-ft)} dk_s dk_g df. \tag{5.7}$$

This motivates the definitions

$$k_x = k_s + k_g \quad \text{and} \quad k_h = k_s - k_g \tag{5.8}$$

or the inverse relationship

$$k_s = \frac{k_x + k_h}{2} \quad \text{and} \quad k_g = \frac{k_x - k_h}{2}. \tag{5.9}$$

This allows equation 5.7 to be written

$$\psi(x,h,t) = \frac{1}{2} \int\limits_{\mathbf{V}_\infty} \phi(k_x,k_h,f) e^{2\pi i(k_x x + k_h h - ft)} dk_x dk_h df. \tag{5.10}$$

where the factor of $1/2$ comes from the *Jacobian* of the coordinate transformation given in equation 5.9. Both of equations 5.5 and 5.10 are proper inverse Fourier transforms which shows that the wavenumber relationships given in equations 5.8 and 5.9 are the correct ones corresponding to the (s,g) to (x,h) coordinate transformation.

Quite a bit can be learned from equations 5.9 about how the spectral content of the pre-stack data gets mapped into the post-stack data. The maximum midpoint wavenumber, k_{xmax} comes

from the sum of k_{smax} and k_{gmax}. As will be shown formally later, the maximum wavenumber is directly proportional to horizontal resolution (i.e. the greater the wavenumber, the better the resolution). The meaning of the maximum wavenumbers is that they are the largest wavenumbers that contain signal. They typically are less than the Nyquist wavenumbers but they cannot be greater. Thus $k_{smax} \leq k_{sNyq} = 1/(2\Delta s)$ and $k_{gmax} \leq k_{gNyq} = 1/(2\Delta g)$. It is customary to sample the x coordinate at $\Delta x = .5\Delta g$ so that $k_{xNyq} = 1/(2\Delta x) = 2k_{gNyq}$ which means that the stacking process will generate k_x wavenumbers up to this value from combinations of k_s and k_g according to equation 5.8. However, because spatial antialias filters (source and receiver arrays) are not very effective, it must be expected that wavenumbers higher than Nyquist will be present in both the k_s and k_g spectra. The only way to generate unaliased wavenumbers up to k_{xNyq} in the stacking process is if $\Delta s = \Delta g$ so that $k_{xNyq} = k_{sNyq} + k_{gNyq}$. This means that there must be a shotpoint for every receiver station which is very expensive acquisition. Normal 2D land shooting puts a shotpoint for every n receiver stations where $n \geq 3$. This means that k_x wavenumbers greater than a certain limit will be formed from completely aliased k_s and k_g wavenumbers. This limiting unaliased wavenumber is

$$k_{xlim} = \frac{1}{2\Delta g} + \frac{1}{2\Delta s} = \frac{1}{2\Delta g} + \frac{1}{2n\Delta g} = \frac{n+1}{2n\Delta g}. \tag{5.11}$$

For n=3, $k_{xlim} = \frac{2}{3}k_{xNyq}$ so that shooting every third group will result in a conventional stack with the upper third of the k_x spectrum being completely useless.

Even if $n = 1$, there will still be aliased contributions to k_x if the k_s and k_g spectra are aliased as they normally are. For example, k_{xNyq} can be formed with unaliased data from $k_{sNyq} + k_{gNyq}$ but it can also be formed from the sum of $k_s = 0$ and $k_g = 2k_{gNyq}$ and there are many more such aliased modes. Similarly, wavenumbers less that k_{xlim} can be formed by a wide variety of aliased combinations. Further analysis is helped by having a more physical interpretation for k_s and k_g so that their potential spectral bandwidth can be estimated.

As explained in section 4.9, the apparent velocity of a wave as it moves across a receiver array is given by the ratio of frequency to wavenumber. For a source record (*common source gather*), this has the easy interpretation that

$$\frac{f}{k_g} = \frac{v_0}{\sin\theta_0} \tag{5.12}$$

which means that

$$k_g = \frac{f \sin\theta_0}{v_0}. \tag{5.13}$$

So, if f_{max} is the maximum temporal frequency and $(\sin\theta_0)_{max} = 1$ then

$$k_{gmax} = \frac{f_{max}}{v_{0min}} \tag{5.14}$$

where v_{0min} is the slowest near surface velocity found along the receiver spread. So, to avoid any aliasing of k_g, assuming point receivers, the receiver sampling must be such that $k_{gNyq} \geq k_{gmax}$ which leads to the antialiasing condition

$$\Delta g \leq \frac{v_{0min}}{2f_{max}}. \tag{5.15}$$

Assuming the equality in expression (5.15) allows the maximal sampling curves in Figure 5.3 to be calculated. Alternatively, if coarser sampling than suggested in these curves is planned, then a portion of the k_g spectrum will be aliased. An array can be designed to suppress the aliased

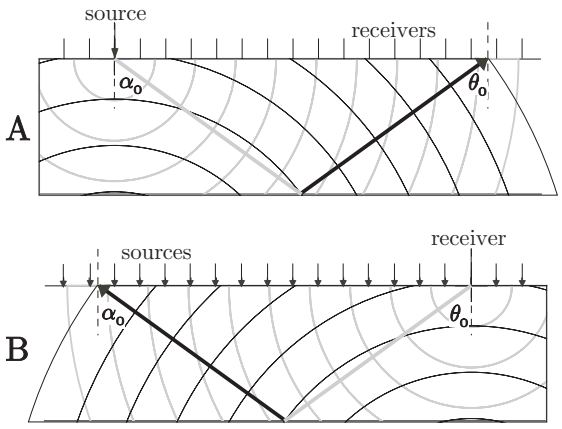

Figure 5.4: (A) Wavenumber k_g is measured on a common source gather and estimates the emergence angle, θ_0, of a monochromatic wave. (B) Wavenumber k_s is measured on a common receiver gather and estimates the takeoff angle α_0 of the ray that arrives at the common receiver.

wavenumbers though this is not as effective as temporal antialias filtering.

The wavenumbers k_s are those which would be measured by an f-k analysis on a common receiver gather. Reciprocity suggests that a common receiver gather is similar to a common source gather with the source and receiver positions interchanged (Figure 5.4). It cannot be expected that the amplitudes will be completely reciprocal on such gathers but the traveltimes should be. Since the relationship between apparent velocity and frequency-wavenumber ratios is a based on traveltimes, not amplitudes, the arguments just made for the interpretation of k_g also apply to k_s. The interpretation is that f/k_s gives the apparent velocity of a monochromatic wave as it leaves a source such that it will arrive at the common receiver. Thus

$$\frac{f}{k_s} = \frac{v_0}{\sin\alpha_0} \tag{5.16}$$

where α_0 is the *take-off angle* required to reach the common receiver and v_0 is as before. Comparing equations 5.12 and 5.16 leads to the conclusion that the potential bandwidth of k_s is just as broad as that of k_g.

For the k_h wavenumber, the stacking process rejects all but $k_h = 0$. Stacking can be modelled by an integral over h as

$$\psi_0(x,t) = \int\limits_{all\ h} \psi(x,h,t)dh. \tag{5.17}$$

If equation (5.10) is substituted into equation (5.17) and the order of integration reversed, it results that

$$\psi_0(x,t) = \frac{1}{2}\int\limits_{\mathbf{V}_\infty} \left[\int\limits_{all\ h} e^{2\pi i k_h h}dh\right] \phi(k_x,k_h,f)e^{2\pi i(k_x x - ft)}dk_x dk_h df. \tag{5.18}$$

The integral in square brackets is $\delta(k_h)$ which, in turn, collapses the k_h integral by evaluating it at

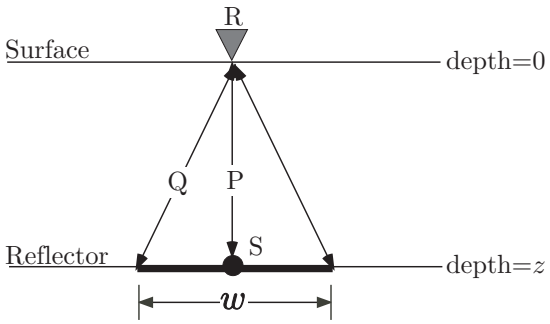

Figure 5.5: The zero-offset *Fresnel zone* is defined as the width of a disk on a subsurface reflector from which scattered energy arrives at R with no more than a $\lambda/2$ phase difference.

$k_h = 0$ to give

$$\psi_0(x,t) = \frac{1}{2}\int\limits_{\mathbf{V}_\infty} \phi(k_x, 0, f) e^{2\pi i(k_x x - ft)} dk_x df. \tag{5.19}$$

So the CMP stacking process passes only the $k_h = 0$ wavenumber which is a very severe f-$k$filter applied to a CMP gather. It is for this reason that f-$k$filtering applied to CMP gathers does not typically improve the stack. However, f-k filtering of source and receiver gathers can have a dramatic effect. The $k_h = 0$ component is the average value over the CMP gather and corresponds to horizontal events. Of course, this is done after normal moveout removal that is designed to flatten those events deemed to be primaries.

So far, the discussion has been about wavenumber spectra with nothing being said about the effect of stacking on temporal frequencies. Bearing in mind the goal to estimate bandlimited reflectivity, the ideal stacked section should have all spectral color removed except that attributable to the reflectivity. Most pre-stack processing flows rely on one or more applications of statistical deconvolution to achieve this. Typically these deconvolution techniques are not capable of distinguishing signal from noise and so whiten (and phase rotate) both. If the pre-stack data input into the stacking process has been spectrally whitened, it will always suffer some attenuation of high frequencies due to the stacking out of noise. This is because the stacking process improves the signal-to-noise ratio by $\sqrt{fold}$. Moreover, this effect will be time-variant in a complicated way because of two competing effects. First, the fold in any stacked section is actually time-variant because of the front-end mute. Until the time at which nominal fold is reached, the stacking effect is therefore time variant. Second, the signal-to-noise ratio in any single pre-stack trace must decrease with both increasing time and increasing frequency due to attenuation (i.e. Q effects). As a result, it is virtually guaranteed that the post-stack data will need further spectral whitening. It is also highly likely that some sort of wavelet processing will be required to remove residual phase rotations.

5.1.4 The Fresnel zone

The zero-offset rays shown in Figure 5.2 are only a high frequency approximation to what really happens. The zero-offset recording at a point on the earth's surface actually contains scattered waves from a very broad zone on each subsurface reflector. This contrasts with the high-frequency raytracing concepts that suggest the reflection occurs at a point. Figure 5.5 illustrates the concept of the zero-offset Fresnel zone. The width, w of the Fresnel zone is defined as the diameter of a

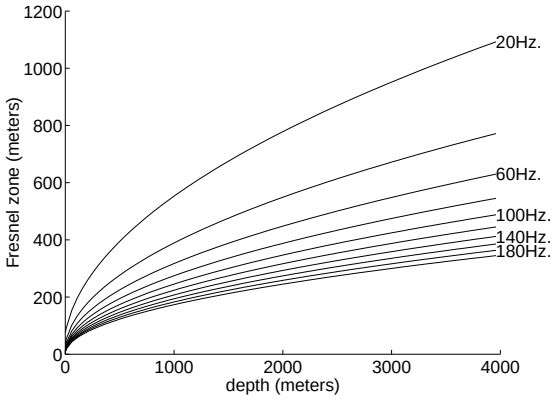

Figure 5.6: The width of the Fresnel zone (based on equation (5.21)) is shown versus depth for a variety of frequencies and an assumed velocity of 3000 m/s.

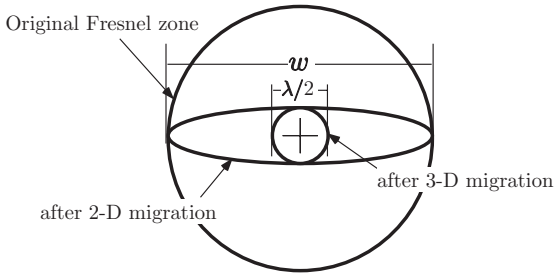

Figure 5.7: A 3D migration collapses the Fresnel zone to a disk of diameter $\lambda/2$ while a 2D migration only collapses the Fresnel zone in the inline direction

disk whose center is marked by the ray P and edge by the ray Q. Ray P is a simple zero-offset ray while ray Q is defined to be $\lambda/4$ longer than ray P. Assuming constant velocity, this means that scattered energy that travels down-and-back along path Q will be $\lambda/2$ out of phase at the receiver, R, compared with that which travels along path P. All paths intermediate to P and Q will show a lesser phase difference. Therefore, it is expected that path Q will interfere destructively with P and is taken to define the diameter of the central disk from which scattered energy shows constructive interference at R. The equation defining the Fresnel zone diameter is therefore

$$\sqrt{z^2 + (w/2)^2} - z = \frac{\lambda}{4} \tag{5.20}$$

which is easily solved for w to give

$$w = 2\sqrt{(\lambda/4 + z)^2 - z^2} = \sqrt{2z\lambda}\sqrt{1 + \lambda/(8z)} = \sqrt{2zv/f}\sqrt{1 + v/(8fz)} \tag{5.21}$$

where, in the third expression, $\lambda f = v$, has been used. If $z \gg \lambda$, then the approximate form $w \sim \sqrt{2z\lambda}$ is often used. The size of a Fresnel zone can be very significant in exploration as it is often larger than the target (Figure 5.6).

Migration can be conceptualized as lowering the source and receiver to the reflector and so shrinks the Fresnel zone to its theoretical minimum of $\lambda/2$. That the Fresnel zone does not shrink to zero is another way of understanding why reflectivity estimates are always bandlimited. However, seismic data is a bit more complicated than this in that it possesses signal over a bandwidth while the Fresnel zone concept is monochromatic. Roughly speaking, the effective Fresnel zone for broadband data can be taken as the average of the individual Fresnel zones for each frequency.

The 3D Fresnel disk collapses, under 3D migration, from a diameter of w to a diameter of $\lambda/2$. However, much seismic data is collected along lines so that only 2D migration is possible. In this case, The Fresnel zone only collapses in the inline direction and remains at its unmigrated size in the crossline direction (Figure 5.7). Thus the reflectivity shown on a 2D migrated seismic line must be viewed as a spatial average in the crossline direction over the width of the Fresnel zone. This is one of the most compelling justifications for 3D imaging techniques, even in the case of sedimentary

basins with flat-lying structures.

5.2 Fundamental migration concepts

5.2.1 One dimensional time-depth conversions

It is quite common to convert a single trace from time to depth, or the reverse, by a simple one dimensional conversion that is called a *stretch*. This is accomplished as a simple mapping of samples from the time trace to the depth trace. The mapping from time to depth is defined by

$$z(\tau) = \int_0^\tau v_{ins}(\tau')d\tau' \tag{5.22}$$

and from depth to time by

$$\tau(z) = \int_0^z \frac{dz'}{v_{ins}(z')} \tag{5.23}$$

where v_{ins} is the *instantaneous* velocity or, equivalently, the local wave speed.

Though not migrations, these one-dimensional operations are very useful for converting from migrated depth to migrated time (or the reverse) or from unmigrated time to unmigrated depth (or the reverse). Thus it is possible to have time or depth displays both before and after migration and the interpreter must be careful to understand what is being presented.

A form of aliasing is possible in all such operations (including migrations) though it is most easily analyzed in one dimension. Consider the conversion of a trace $s(t)$ to a depth trace $\hat{s}(z)$ with a desired depth sample interval of Δz. The depth trace will usually be constructed by a loop over the desired depth samples. Assuming constant velocity for simplicity, a particular depth sample of $\hat{s}(z)$ at $z = n\Delta z$ must be interpolated from $s(t)$ at $t = 2n\Delta z/v$. Thus, the time to depth conversion will effectively extract samples from $s(t)$ at the interval $\Delta t_{eff} = 2\Delta z/v$. This is a form of resampling. If $\Delta t_{eff} > \Delta t$ then aliasing will occur as the depth trace is constructed unless an antialias filter is applied. If f_{max} is the maximum signal frequency (presumed to be less than the Nyquist frequency) in $s(t)$, then the *sampling theorem* (Karl, 1989) assures that $s(t)$ can be resampled to $\Delta t \leq 1/(2f_{max})$ (with antialias filtering to prevent noise aliasing) without loss of signal. Therefore, the depth sample interval should not be chosen arbitrarily but should be chosen to ensure that Δt_{eff} preserves signal. This leads to the condition

$$\Delta z \leq \frac{v_{min}}{4f_{max}}. \tag{5.24}$$

In this expression, v_{min} is used to ensure that all signal frequencies are preserved in the worst case.

These considerations are true for multi-dimensional migrations as well as simple stretching. The depth sample interval should be chosen with knowledge of the maximum signal frequency. Furthermore the temporal data should be highcut filtered to reject frequencies larger than the maximum frequency used to compute the depth sample interval.

5.2.2 Raytrace migration of normal-incidence seismograms

Most modern migration techniques are based on wavefield concepts that treat the recorded data as a boundary value. This was not always the case; however, because raytrace migration was very popular before 1980. As is often true, raytrace methods provide a great deal of insight into the problem and can be adapted to almost any setting. Understanding raytrace migration can help considerably in understanding the corresponding wavefield technique. For example, the very word “migration”

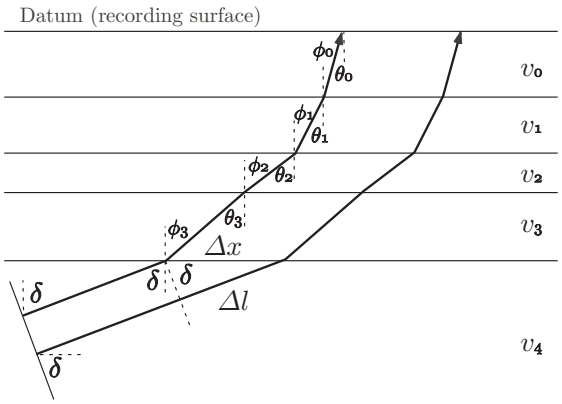

Figure 5.8: Two normal-incidence rays from a dipping reflector beneath a $v(z)$ medium. The delay between the rays allows the ray parameter to be estimated.

comes from a popular, though somewhat incorrect, view that the process just "moves events around" on a seismic section. This encourages a fundamental, and very pervasive, confusion that time and depth sections are somehow equivalent. A better view is that an unmigrated time section and a migrated depth section are independent (in fact orthogonal) representations of a wavefield. Raytrace migration emphasizes this latter view and makes the role of the velocity model very clear.

Figure 5.8 illustrates the link between reflector dip and the measurable traveltime gradient (for normal-incidence rays) on a ZOS. The figure shows two rays that have normal incidence on a dipping reflector beneath a $v(z)$ medium. More general media are also easily handled, this just simplifies the discussion. The two rays are identical except for an extra segment for the lower ray in the bottom layer. From the geometry, this extra segment has a length $\Delta x \sin\delta$ where Δx is the horizontal distance between the rays. If there were receivers on the interface between layers three and four, then for the upcoming rays, they would measure a traveltime delay, from left to right, of

$$\Delta t = \frac{2\Delta x \sin\delta}{v_4} \tag{5.25}$$

where the factor of two accounts for two-way travel. This implies a horizontal traveltime gradient (horizontal slowness) of

$$\frac{\Delta t}{\Delta x} = 2\frac{\sin\delta}{v_4}. \tag{5.26}$$

Inspection of the geometry of Figure 5.8 shows that equation (5.26) can be rewritten as

$$\frac{\Delta t}{\Delta x} = 2p_n \tag{5.27}$$

where p_n is the ray parameter of the normal-incidence ray. In the $v(z)$ medium above the reflector, Snell's law ensures that p_n will remain equal to twice the horizontal slowness at all interfaces including the surface at $z = 0$. Thus, a measurement of $\Delta t/\Delta x$ at the surface of a *normal-incidence seismogram* completely specifies the raypaths provided that the velocity model is also known. In fact, using equation (5.26) allows an immediate calculation of the reflector's dip.

Of course, in addition to the reflector dip, the spatial position's of the reflection points of the normal rays are also required. Also, it must be expected that more general velocity distributions

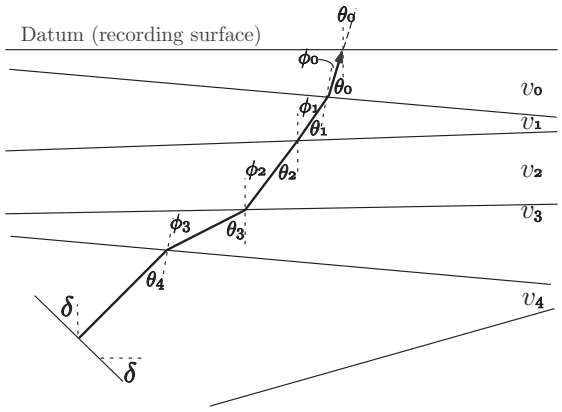

Figure 5.9: The migration of a normal-incidence ray is shown. Knowledge of the ray parameter allows the ray to be traced down through the velocity model.

than $v(z)$ will be encountered. In $v(x, y, z)$ media, the link between reflector dip and horizontal slowness is not as direct but knowledge of one still determines the other. *Normal-incidence raytrace migration* provides a general algorithm that handles these complications. This migration algorithm will be described here for 2D but is easily generalized to 3D. Before presenting the algorithm, a formal definition of a *pick* is helpful:

> A *pick* is defined to be a triplet of values $(x_0, t_0, \Delta t/\Delta x)$measured from a ZOS having the following meaning
>
> x ... inline coordinate at which the measurement is made.
>
> t_0... normal-incidence traveltime (two way) measured at x.
>
> $\Delta t/\Delta x$ ... horizontal gradient of normal-incidence traveltime measured at (x, t_0).

To perform a raytrace migration, the following materials are required:

1. $(x_i, t_{0ij}, \Delta t/\Delta x_{ij})$ ... a set of picks to be migrated. The picks are made at a the locations x_i and at each location a number of t_{0ij} and $\Delta t/\Delta x_{ij}$ values are measured.
2. $v(x, z)$ ... a velocity model is required. It must be defined for the entire range of coordinates that will be encountered by the rays.
3. Computation aids... one or more of: a calculator, compass, protractor, computer.

Finally, the algorithm is presented as a series of steps with reference to Figure 5.9. For all locations $x_i, i = 1, 2, \ldots n$ then each ray must be taken through the following steps:

Step 1 ... Determine the emergence angle at the surface of the ij$^{\text{th}}$ ray according to the formula

$$\sin\theta_{0ij} = \frac{v_0(x_i)}{2}\frac{\Delta t}{\Delta x}_{ij} \tag{5.28}$$

Step 2 ... Denote the current layer by the number h. Project the ray down to the next interface and determine the incident angle ϕ_{hij}.

Step 3 ... Determine the length of the ij$^{\text{th}}$ raypath in the current (h$^{\text{th}}$) layer, l_{hij} and compute the layer traveltime

$$\delta t_{hij} = \frac{l_{hij}}{v_h} \tag{5.29}$$

Step 4 ... Determine the refraction angle into the $(h+1)^{\text{th}}$ layer from Snell's law:

$$\sin(\theta_{(h+1)ij}) = \frac{v_{h+1}}{v_h} \sin(\phi_{hij}) \tag{5.30}$$

Step 5 ... Repeat steps 2 $\rightarrow$ 4 for each velocity layer until the stopping condition is fulfilled

$$t_{ij} = \sum_{h=1}^{n} \delta t_{hij} = \frac{t_{0ij}}{2} \tag{5.31}$$

which simply says that the raypath traveltime must be half the measured traveltime.

Step 6 ... Draw in a reflecting segment perpendicular to the raypath at the point that the stopping condition is satisfied.

5.2.3 Time and depth migration via raytracing

When the first wavefield migration methods were being developed, the finite-difference approach was pioneered by Jon Claerbout (Claerbout, 1976) and his students at Stanford University. This very innovative work produced dramatic improvements in the quality of seismic images but the algorithms were a strong challenge for the available computers. As a result approximations were sought which could reduce the numerical effort required. A key approximation was Claerbout's use of a moving coordinate system to derive an approximate wave equation that essentially substituted vertical traveltime, τ, for depth. When the finite difference machinery was implemented with this approximate equation the resulting migrated images had a vertical coordinate of time not depth. It also resulted, for reasons to be discussed when finite-difference methods are presented, in much faster run times. One way to see why this might be is to imagine a stacked section consisting only of events with $\Delta t/\Delta x = 0$. This is not far from the appearance of seismic sections from many of the world's sedimentary basins. Such basins are good $v(z)$ environments and the normal rays for flat events are all vertical. Thus, they are already positioned correctly in vertical traveltime on the stacked section and the migration algorithm has very little to do. This is a drastic over simplification that will be more correctly stated later, but it comes close to the truth.

These early finite-difference methods were used with great success in the sedimentary basins around the world. However, when the technology was taken into thrust belts, continental margins, and other areas where the $v(z)$ approximation is not a good model, it was gradually realized that systematic imaging errors were occurring. Today, the reason is well understood and it was a consequence of the approximate wave equation discussed in the previous paragraph. Figure 5.10 suggests the problem. Here an anticline is shown positioned beneath a slower-velocity near-surface that has a dipping bottom interface. The normal ray from the crest of the structure is generally not the ray of minimum traveltime even though it has the shortest path to the surface. Instead, a ray from a point to the right is the least-time ray because it spends more of its path in the fast material. Thus, on a normal incidence section, the traveltime signature of the anticline will have a crest at the emergence

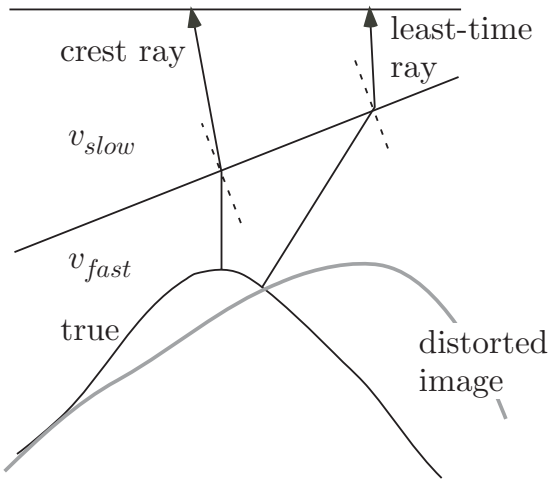

Figure 5.10: The least-time ray does not always come from the crest of an anticline.

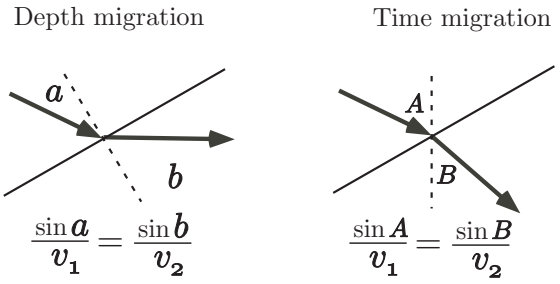

Figure 5.11: (Left) When a ray encounters a dipping velocity interface, Snell's law predicts the transmitted ray using a relation between angles with-respect-to the interface normal. (Right) In a time migration algorithm, Snell's law is systematically violated because the vertical angles are used. Both techniques give the same result if the velocity interface is horizontal.

point of the least-time ray. Generally, a traveltime crest will have zero $\Delta t/\Delta x$ which means that the corresponding ray emerges vertically. It turned out that the migration schemes were producing a distorted image in cases like this and the distortion placed the crest of the migrated anticline at the emergence point of the least time ray. Just like in the sedimentary basin case, the algorithm was leaving flat events alone even though, in this case, it should not.

The understanding and resolution of this problem resulted in the terms *time migration* and *depth migration*. The former refers to any method that has the bias towards flat events and that works correctly in $v(z)$ settings. The latter refers to newer methods that were developed to overcome these problems and therefore that can produce correct images even in strong lateral velocity gradients. Robinson (1983) presented the raytrace migration analogs to both time and depth migration and these provide a great deal of insight. The raytrace migration algorithm presented previously in section 5.2.2 is a depth migration algorithm because it obeys Snell's law even when $\partial_x v$ is significant.

Figure 5.11 shows how Snell's law must be systematically altered if a time migration is desired. Instead of using the angles that the incident and transmitted rays make with respect to the normal

in Snell's law, the time migration algorithm uses the angles that the rays make with respect to the vertical. It is as though the dipping interface is rotated locally to the horizontal at the incident point of the ray. Thus, any vertical ray (i.e. $\Delta t/\Delta x = 0$) will pass through the velocity interface without deflection regardless of the magnitude of the velocity contrast.

This explains why time migration gets the wrong answer and it also explains why the technology remains popular despite this now well-understood shortcoming. The fact that flat events are unaffected by time migration regardless of the velocity model means that the process has a "forgiving" behavior when the velocity model has errors. Velocity models derived automatically from stacking velocities tend to be full of inaccuracies. When used with time migration, the inaccuracies have little effect but when used with depth migration they can be disastrous. Even today, the majority of seismic sections come from $v(z)$ settings and they consist of mostly flat events. Such data can be migrated with relatively little care using time migration and well-focused sections are obtained. Of course, these are still time sections and a pleasing time migration does not mean that the migration velocities can be used for depth conversion. In general, the velocity errors that are "ignored" by time migration will cause major problems in a subsequent depth conversion. Much more care must be taken if an accurate depth section is required.

The terms *time migration* and *depth migration* arose, in part, because the early versions of both technologies tended to produce only time and depth sections respectively. However, this is no longer the case and either technology can produce both time and depth sections. Therefore, the terms should be treated simply as jargon that indicates whether or not an algorithm is capable of producing a correct image in the presence of strong lateral velocity gradients. The mere presence of a time or depth axis on a migrated seismic display says nothing about which technology was used. In this book, time migration will be used to refer to any migration technique that is strictly valid only for $v(z)$ while depth migration will refer to a technique that is valid for $v(x, z)$. Thus, when employed in constant velocity or $v(z)$ settings, both techniques are valid and there is no meaningful distinction. Under these conditions, a migrated time section can always be converted to a depth section (or vice-versa) with complete fidelity.

The first time-migration algorithms were almost always finite-difference methods. Today, a better understanding of the subject allows virtually any migration method to be recast as a time migration. Therefore, it is very important to have some knowledge of the algorithm, or to have a statement from the developer, to be sure of the nature of a method. The issue has been clouded even further by the emergence of intermediate techniques. Sometimes, it is worth the effort to build a synthetic seismic section to test a migration algorithm whose abilities are uncertain. One thing remains clear, depth migrations always require much more effort for a successful result. Not only is more computer time required; but, much more significantly, vastly greater human time may be necessary to build the velocity model.

5.2.4 Elementary wavefront techniques

Though very versatile, the raytrace method described in section 5.2.2 has its limitations. For example, it is not obvious how to make the migrated amplitudes any different from the unmigrated ones and it is equally obvious that amplitudes should change under migration. Recall that bandlimited reflectivity is the goal and that the geometrical spreading of wavefronts must be accounted for to estimate this. (See section 5.1.1.) Wavefront methods provide amplitude adjustments in a very natural way though they are more difficult to adapt to complex media than the raytrace approach.

Hagedoorn (1954) provided one of the first, and still very relevant, explanations of wavefront migration. The ideas presented in this section are all essentially attributable to him. Consider the ZOS image of a dipping reflector overlain by a constant velocity medium as shown in Figure 5.12A.

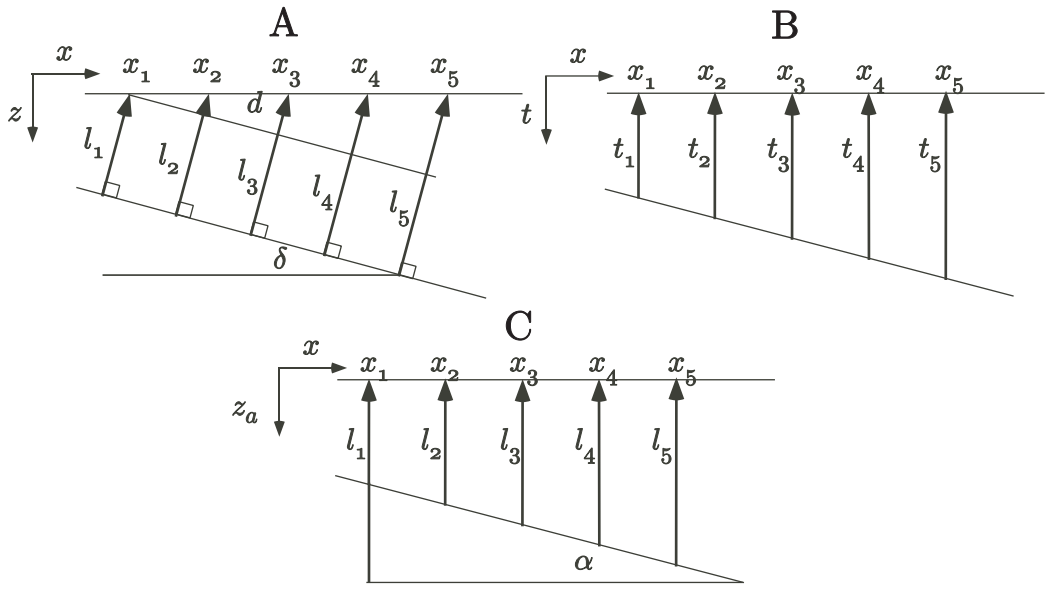

Figure 5.12: (A) Normal-incidence raypaths are shown for a dipping reflector in a constant velocity medium. (B) The traveltime section for (A) plots the arrival times for each raypath beneath its emergence point. (C) When (B) is plotted at apparent depth, $z_a = vt/2$, an apparent dip can be defined.

The reflector dip, δ, can be related to the lengths of the raypaths and their emergence points by

$$\sin\delta = \frac{l_5 - l_1}{x_5 - x_1}. \tag{5.32}$$

The time section corresponding to this raypath diagram is shown in Figure 5.12B. Since velocity is constant, the arrival times are directly proportional to the path lengths through $t_k = 2l_k/v$ and the ZOS image of the reflector is seen to have a time dip of

$$\frac{\Delta t}{\Delta x} = \frac{t_5 - t_1}{x_5 - x_1} = \frac{2}{v}\frac{l_5 - l_1}{x_5 - x_1} = 2\frac{\sin\delta}{v} \tag{5.33}$$

which is in agreement with equation (5.27) that the time dip on a ZOS gives twice the normal-incidence ray parameter. A further conceptual step can be taken by mapping the traveltimes to *apparent depth*, z_a, by $z_a = vt/2$ as shown in Figure 5.12C. At this stage, the unmigrated display has both coordinate axes in distance units so that it makes sense to define an *apparent dip* through

$$\tan\alpha = \frac{l_5 - l_1}{x_5 - x_1}. \tag{5.34}$$

Finally, comparison of equations (5.32) and (5.34) gives the relation

$$\sin\delta = \tan\alpha. \tag{5.35}$$

Equation (5.35) is called the *migrator's equation* and is of conceptual value because is illustrates the geometric link between unmigrated and migrated dips. However, it should not be taken too far. It is not generally valid for non-constant velocity and cannot be applied to a time section. Though this last point may seem obvious, it is often overlooked. It is not meaningful to talk about a dip in degrees for an event measured on a time section because the axes have different units. A change of time-scale changes the apparent dip. On an apparent depth section, an apparent dip can be meaningfully defined and related to the true dip through equation (5.35). The apparent dip can never be larger than 45° because $\sin\delta \leq 1$ which is a re-statement of the fact that the maximum time dip is $\Delta t/\Delta x = 1/v$ as discussed in section 4.11.1. Though intuitively useful, the apparent dip concept is not easily extended to variable velocity. In contrast, the limitation on maximum time dip is simply restated in $v(z)$ media as $\Delta t/\Delta x = 1/v_{min}$ where v_{min} is the minimum of $v(z)$.

Comparison of Figures 5.12A and 5.12C shows that, for a given surface location x_k, the normal-incidence reflection point and the corresponding point on the ZOS image both lie on a circle of radius l_k centered at x_k. This relationship is shown in Figure 5.13 from which it is also apparent that

- The ZOS image lies at the nadir (lowest point) of the circle.
- The reflection point is tangent to the circle.
- The ZOS image and the reflector coincide on the datum (recording plane).

These observations form the basis of a simple wavefront migration technique which is kinematically exact for constant velocity:

i) Stretch (1D time-depth conversion) the ZOS image from t to z_a.

ii) Replace each point in the (x, z_a) picture with a wavefront circle of radius z_a. Amplitudes along the wavefront circle are constant and equal to the amplitude at (x, z_a).

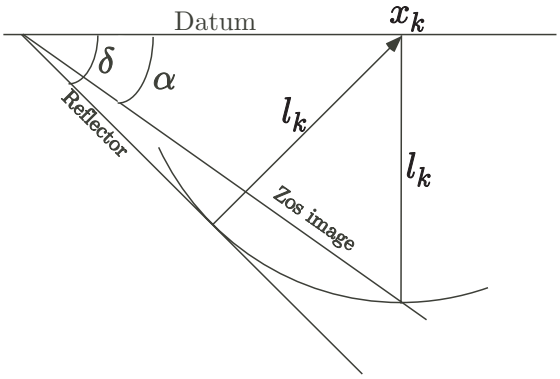

Figure 5.13: The normal-incidence reflection point and the corresponding point on the ZOS image both lie on a circle of radius l_k centered at x_k.

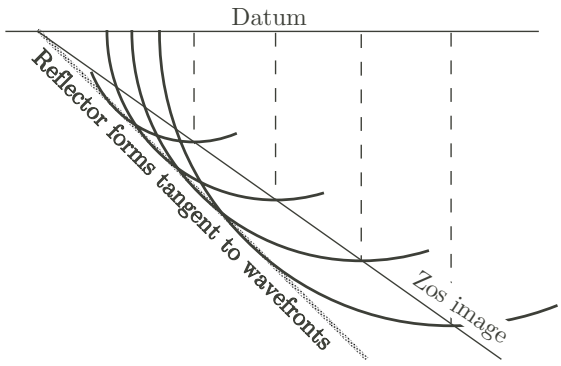

Figure 5.14: The wavefront migration of a dipping reflector is shown. Each point on the ZOS image is replaced by a wavefront circle of radius l_k. The wavefronts interfere constructively at the location of the actual reflector.

iii) The superposition of all such wavefronts will form the desired depth section.

Figure 5.14 shows the migration of a dipping reflector by wavefront superposition. The actual mechanics of the construction are very similar to a convolution. Each point in the ZOS image is used to scale a wavefront circle which then replaces the point. A 2D convolution can be constructed by a similar process of replacement. The difference is that the replacement curve in a 2D convolution is the same for all points while in the migration procedure it varies with depth. Thus the migration process can be viewed as a *nonstationary* convolution.

Wavefront migration shows that the *impulse response* of migration is a wavefront circle. That is, if the input ZOS contains only a single non-zero point, then the migrated section will be a wavefront circle. The circle is the locus of all points in (x, z) that have the same traveltime to $(x_k, 0)$. Two equivalent interpretations of this are either that the circle is the only earth model that could produce a single output point or that the circle is the curve of *equal probability*. The first interpretation is a deterministic statement that only one earth model could produce a ZOS image with a single live sample. The second interpretation is a stochastic one that suggests why the generalization from one live sample to many works. By replacing each point with it's curve of equal probability, the migrated section emerges as the most likely geology for the superposition of all points. The constructive interference of a great many wavefronts forms the migrated image.

Exercise 5.2.1. *Describe the appearance of:*

- *a migrated section that contains a high amplitude noise burst on a single trace.*
- *a migration that results from a ZOS image whose noise level increases with time.*
- *the edge of a migrated section.*

5.2.5 Huygens' principle and point diffractors

Christian Huygens was an early physicist and astronomer (a contemporary of Newton) who made a number of advances in the understanding of waves. Most notable was his principle that, if the

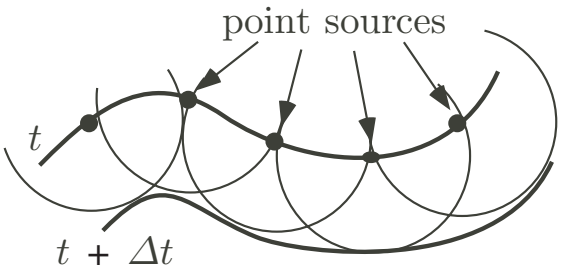

Figure 5.15: Huygens' principle reconstructs the wavefront at $t + \Delta t$ by the superposition of the small wavefronts from secondary point sources placed on the wavefront at t.

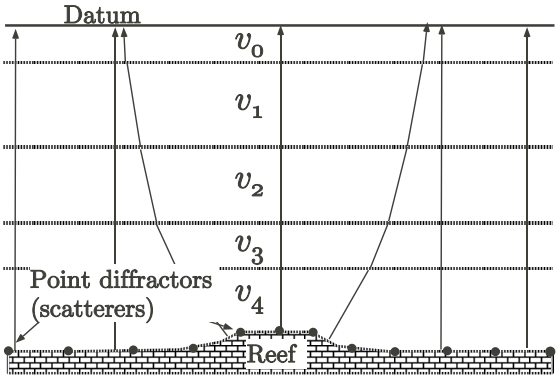

Figure 5.16: The seismic response of a continuous geological structure is synthesized as the superposition of the responses of many *point diffractors*.

position of a wavefront at time t is known, its position at $t + \Delta t$ can be computed by a simple strategy. Each point on the wavefront at time t is considered to be a secondary source of a small spherical (3D) or circular (2D) wavefront called a *Huygens' wavelet* as shown in Figure 5.15. For the seismic problem, Huygens' principle can be adapted to consider the response of a continuous reflector as the superposition of the responses of a great many *point diffractors* or *scatterpoints*(Figure 5.16).

Figure 5.17 shows the concept of a point diffractor that scatters any incident wave in all directions. Thus the complete imaging of any reflector is only possible if all of this scattered energy is captured and focused at the scatterpoint. Each scattered raypath is characterized by its scattering angle θ. If the scatterpoint lies on a dipping reflector then the scattered ray that coincides with the normal ray will be the strongest after superposition of many scatterpoints. For this ray the scattering angle is the same as the reflector dip which suggests why the scattering angle is commonly called the "migration dip" in a migration program. However, *migration dip* still conveys a misleading impression to many untutored users who wish to equate it with geologic dip. They draw the mistaken conclusion that, for data with low geologic dips, it suffices to correctly handle only low migration dips. The correct interpretation of migration dip as *scattering angle* shows that even for low geologic dips it is desirable to handle large scattering angles. In fact, a high-resolution migration requires that scattering angle be as large as possible. In the context of section 5.1.3 scattering angle is directly related to spectral components through $k_x/f = .5 \sin\theta/v$ or equivalently $k_x = .5f \sin\theta$. This says that high k_x values require large values of θ. It is a fundamental component of resolution theory that the k_x spectrum must be as large as possible for high resolution.

The traveltime curve of a point diffractor, for zero-offset recording and constant velocity, has a raypath geometry that is essentially equivalent to that for a CMP gather and a zero-dip reflector (Figure 5.18). Recall that the NMO traveltime curve is given by $t_H^2 = t_0^2 + H^2/v^2$ where H is the full source-receiver offset. This means that the traveltime curve for a zero-offset point diffractor is given by

$$t_x^2 = t_0^2 + \frac{4(x - x_0)^2}{v^2} \tag{5.36}$$

where the diffractor is at (x_0, z), $t_0 = 2z/v$, and the velocity, v, is constant. The factor of 4 in the numerator arises because $x - x_0$ corresponds to the half-offset. For $v(z)$, the conclusion is immediate

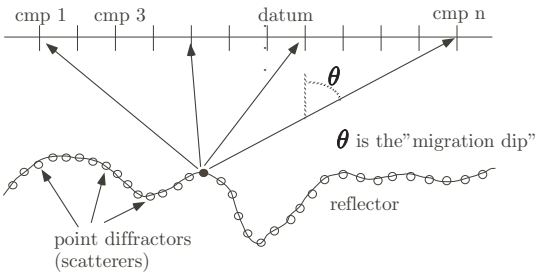

Figure 5.17: Each point on a reflector is considered to be a point diffractor that scatters energy to all surface locations.

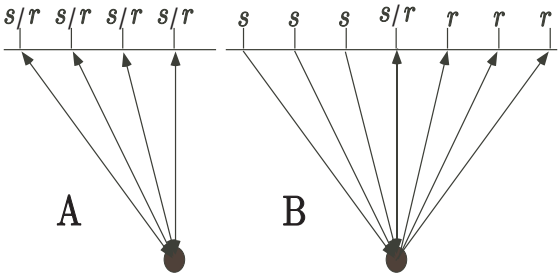

Figure 5.18: (A) The ray paths for the left side of a zero-offset point diffractor. (B) The raypaths for a CMP gather assuming a zero-dip reflector. These raypaths are identical for $v(z)$ leading to the conclusion that the traveltime curves for the point diffractor are the same as for the NMO experiment.

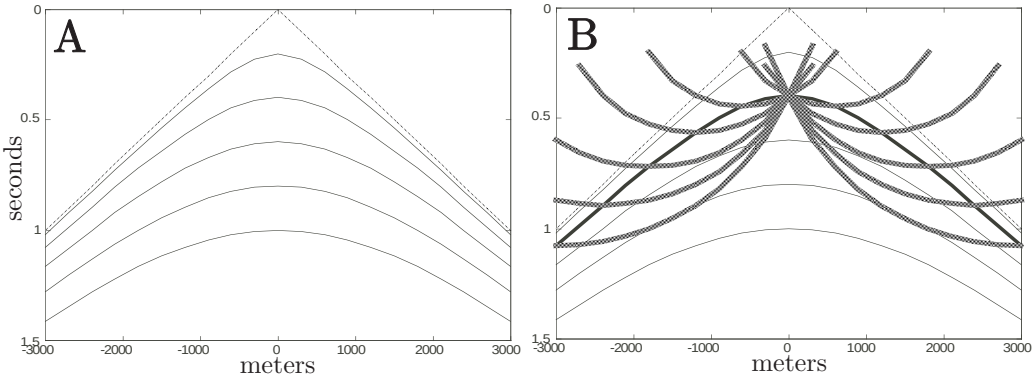

Figure 5.19: (A) This *diffraction chart* shows the traveltime curves for five point diffractors in a constant velocity medium. All five curves are asymptotic to the line x=vt/2. (B) The second hyperbola has been migrated by wavefront superposition.

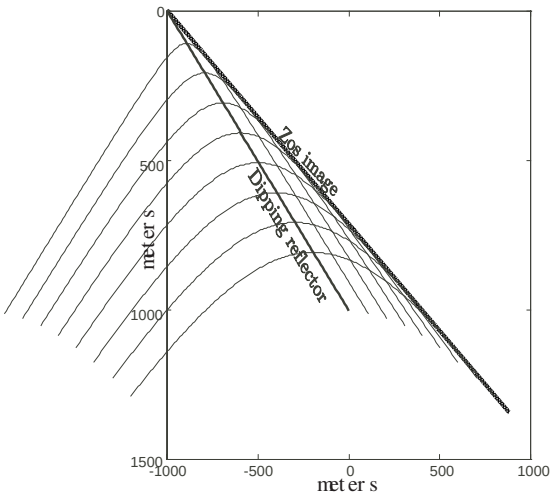

Figure 5.20: The ZOS image of the dipping reflector is formed by replacing each point on the reflector with the ZOS image of a point diffractor. The superposition of these many hyperbolae forms the dipping reflector's ZOS image.

that the Dix equation proof that the NMO hyperbola is approximately characterized by v_{rms} means that the diffraction traveltimes are approximately

$$t_x^2 \approx t_0^2 + \frac{4(x - x_0)^2}{v_{rms}^2} \tag{5.37}$$

where $t_0 = 2z/v_{ave}$.

Figure 5.14 shows how wavefront superposition migrates the ZOS image of a dipping reflector. The ZOS image of a single point diffractor is the hyperbola given by equation (5.36) so it is instructive to see how wavefront migration can collapse such hyperbolae. Figure 5.19A shows a *diffraction chart* that gives the traveltime curves for five point diffractors in a constant velocity medium. The wavefront migration of this chart should convert it into five impulses, one at the apex of each hyperbola. Figure 5.19B shows the wavefront migration of the second hyperbola on the chart. Each point on the second hyperbola has been replaced by a wavefront circle whose radius is the vertical time of the point. Since the chart is drawn with a vertical coordinate of time rather than depth, these wavefront curves may not appear to be exactly circular. The geometry of hyperbola and circles is such that the wavefront circles all pass through the apex of the second hyperbola. Thus the amplitude of the wavefront superposition will be large at the apex and small elsewhere. The same process will also focus all other hyperbolae on the chart simultaneously.

The diffraction curves can be used to perform the inverse of migration or *modelling*. Figure 5.20 shows the construction of the ZOS image of a dipping reflector using Huygens' principle. Each point on the dipping reflector is replaced by the ZOS image of a point diffractor. The superposition of these many hyperbolae constructs the ZOS image of the dipping reflector. For a given hyperbola, its apex lies on the geology and it is tangent to the ZOS image. This directly illustrates that the wavefront migration of dipping reflectors is completely equivalent to the migration of diffraction responses. In fact, the collapse of diffraction responses is a complete statement of the migration problem and a given algorithm can be tested by how well it achieves this goal.

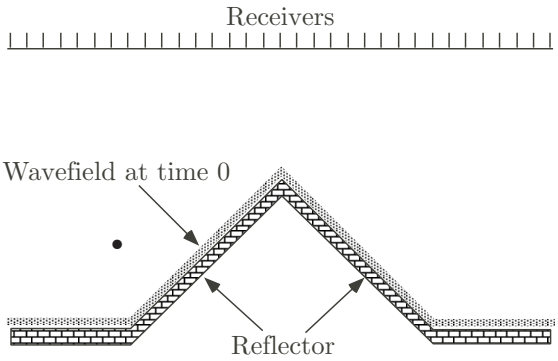

Figure 5.21: The exploding reflector model at the instant of explosion establishes a wavefield that is isomorphic with the reflector.

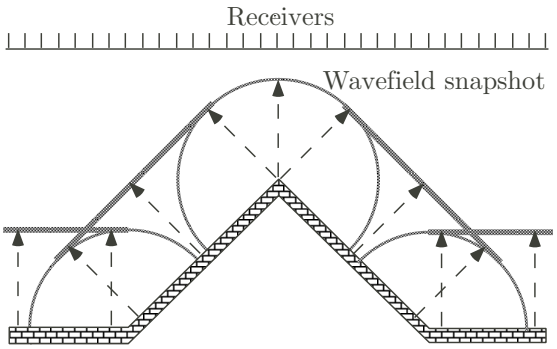

Figure 5.22: A snapshot of the ERM wavefield at some instant of time as it approaches the receivers

5.2.6 The exploding reflector model

In the preceding sections, two methods have been described to migrate stacked data. In section 5.2.2 the stacked section was treated as a normal-incidence seismogram and raytrace techniques were developed to migrate it. In section 5.2.4 the wavefront migration method was presented for constant-velocity zero-offset sections. Further progress is greatly facilitated by a more abstract model of the stacked section. Ultimately, the goal is to formulate the post-stack migration problem as a solution to the wave equation where the measured data plays the role of a *boundary value*. However, this is complicated by the fundamental fact that the CMP stack is not a single wavefield but the composite of many individual wavefields. There is no single physical experiment that could record a ZOS and so a ZOS cannot be a single physical wavefield.

There is a useful thought experiment, called the *exploding reflector model* (ERM), that does yield something very similar to a ZOS and serves as the basis for most post-stack migration methods. (The following discussion is presented in 2D and the generalization to 3D is elementary.) As motivation, note that equation (5.36) can be rewritten as

$$t_x^2 = t_0^2 + \frac{(x - x_0)^2}{\hat{v}^2} \tag{5.38}$$

where $\hat{v} = v/2$ is called the *exploding reflector velocity* and $t_0 = z/\hat{v}$. This trivial recasting allows the interpretation that the point diffractor is a seismic source of waves that travel at one-half of the physical velocity. As shown in Figure 5.21 the exploding reflector model adopts this view and postulates a model identical to the real earth in all respects except that the reflectors are primed with explosives and the seismic wave speeds are all halved. Receivers are placed at the CMP locations and at $t = 0$ the explosives are detonated. This creates a wavefield that is morphologically identical to the geology at the instant of explosion.

If $\psi(x, z, t)$ denotes the exploding reflector wavefield, then the mathematical goal of the migration problem is to calculate $\psi(x, z, t = 0)$. Thus $\psi(x, z, t = 0)$ is identified with the geology and represents the migrated depth section. The ERM wavefield is allowed to propagate only upward (the $-z$ direction) without reflections, mode conversions, or multiples. It refracts according to Snell's law and the half-velocities mean that the traveltimes measured at the receivers are identical to those of the ZOS. In the constant velocity simulation of Figure 5.22, a *snapshot* of the ERM wavefield, $\psi(x, z, t =$

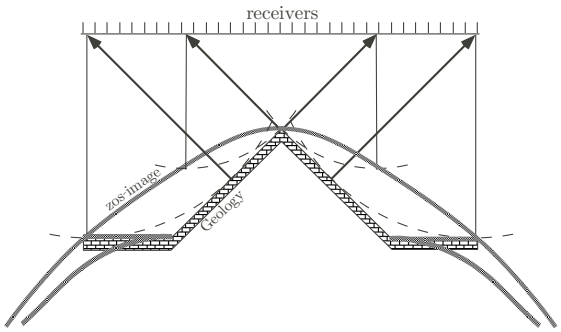

Figure 5.23: The ERM seismogram is shown superimposed on the migrated depth section.

const), is shown as the wavefield approaches the receivers. The raypaths are normal-incidence raypaths (because of the isomorphism between the geology and $\psi(x, z, t = 0)$) and spreading and buried foci are manifest. The figure emphasizes three Huygen's wavelets that emanate from the three corners of the geologic model. The migrated depth section is also a snapshot, but a very particular one, so the term *snapshot* will be used to denote all such constant-time views.

In Figure 5.23, the ERM seismogram, $\psi(x, z = 0, t)$, is shown in apparent depth superimposed on top of the depth section. Wavefront circles are shown connecting points on the geology with points on the seismogram. The ERM seismogram is kinematically identical (i.e. the traveltimes are the same) with the normal-incidence seismogram and is thus a model of the CMP stack. It is also kinematically identical with all normal-incidence, primary, events on a ZOS image. This allows the migration problem to be formulated as a solution to the wave equation. Given $\psi(x, z = 0, t)$ as a boundary condition, a migration algorithm solves the wave equation for the entire ERM wavefield, $\psi(x, z, t)$, and then sets $t = 0$ to obtain the migrated depth section. This last step of setting $t = 0$ is sometimes called *imaging* though this term has also come to refer to the broader context of migration in general. Precisely how the wavefield is evaluated to extract the geology is called an *imaging condition*. In the post-stack case the imaging condition is a simple evaluation at $t = 0$. Later, it will be seen that there are other imaging conditions for pre-stack migration.

Thus far, the ERM has given simple mathematical definitions to the migrated depth section, the recorded seismogram, and the wavefield snapshot. In addition, the *extrapolated seismogram* can be defined as $\psi(x, z = \Delta z, t)$. Both the ERM seismogram and the extrapolated seismogram are time sections while the snapshot and the migrated section are in depth. The extrapolated seismogram is a mathematical simulation of what would have been recorded had the receivers been at $z = \Delta z$ rather than at $z = 0$. The construction of $\psi(x, z = \Delta z, t)$ from $\psi(x, z = 0, t)$ is called *downward continuation* and, synonymously, wavefield extrapolation.

As a summary, the ERM has defined the following quantities:

$\psi(x, z, t)$ ... the ERM wavefield.

$\psi(x, z, t = 0)$ ... the migrated depth section.

$\psi(x, z, t = const)$ ... a wavefield snapshot.

$\psi(x, z = 0, t)$ ... the ERM seismogram.

$\psi(z, z = \Delta z, t)$ ... an extrapolated section.

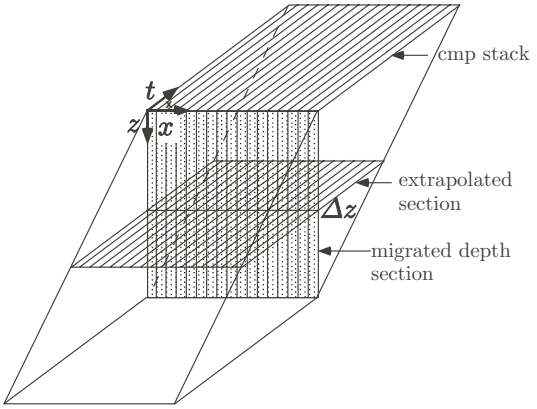

Figure 5.24: This prism shows the portion of (x, z, t) space that can be constructed from a finite-extent measurement of $\psi(x, z = 0, t)$ (upper face). Also shown are the migrated depth section, $\psi(x, z, t = 0)$, (vertical slice) and an extrapolated section, $\psi(x, z = \Delta z, t)$, (horizontal slice).

These quantities are depicted in Figure 5.24. It is significant that any extrapolated seismogram can be evaluated at $t = 0$ to give a single depth sample of the migrated depth section. The process of deducing a succession of extrapolated seismograms, and evaluating each at $t = 0$, is called a *recursive migration*. It is recursive because the extrapolated seismogram $\psi(x, z = z_k, t)$ is computed from the previous seismogram $\psi(x, z = z_{k-1}, t)$. Examples of recursive migrations are the finite-difference methods and the phase-shift techniques. An alternative to the recursive approach is a *direct migration* that computes $\psi(x, z, t = 0)$ directly from $\psi(x, z = 0, t)$ without the construction of any intermediate products. Examples of direct migration are (k_x, f) migration and Kirchhoff migration.

A direct migration tends to be computationally more efficient than a recursive approach both in terms of memory usage and computation speed. However, the direct methods must deal with the entire complexity of the velocity structure all at once. In contrast, the recursive approach forms a natural partitioning of the migration process into a series of wavefield extrapolations. The extrapolation from $z = z_{k-1}$ to $z = z_k$ need only deal with the velocity complexities between these two depths. If the interval $z_{k-1} \rightarrow z_k$ is taken sufficiently small, then the vertical variation of velocity can be ignored and v can be considered to depend only upon the lateral coordinates. In this way the migration for a complex $v(x, z)$ variation can be built from the solution of many $v(x)$ problems.

5.3 MATLAB facilities for simple modelling and raytrace migration

This section describes three facilities that can produce simple seismic section models. The first two, hyperbolic superposition and finite differencing, create full waveform models suitable for migration with full waveform migration methods. The third, normal incidence raytracing, merely predicts arrival times of events and not their amplitudes.

5.3.1 Modelling by hyperbolic superposition

Section 5.2.5 describes how the seismic response of a complex geological structure can be viewed as the superposition of the responses of many point diffractors. Conceptually, infinitely many point diffractors are required and the seismic response is found in the limit as the number of diffractors becomes infinite and their spacing infinitesimal. In a practical computer implementation, only a finite number of diffractors can be simulated and their spacing can be no finer than the underlying computation grid.

There are six basic routines that support modelling by superposition of hyperbolae. These are collected in the *synsections* toolbox and assume that a constant-velocity, zero-offset synthetic is desired. The fundamental paradigm is that each routine inserts a single event in a matrix that represents the seismic section. These six basic commands are

event_spike Inserts an isolated noise spike.

event_hyp Inserts a hyperbolic event.

event_dip Inserts a dipping (linear) event.

event_diph Builds a dipping event by superimposing hyperbolae.

event_diph2 Similar to *event_diph* plus it allows control over the hyperbolae spacing.

event_pwlinh Superimposes hyperbolae along a piecewise linear track.

These all operate similarly and insert a spike, hyperbola, linear event, or piecewise linear event in a matrix. The matrix must be created externally and assigned vertical (time) and horizontal (distance) coordinates. Then the inserted event is described by its position in (x, t) or (x, z). Some of the commands require the specification of velocity and this should be the physical velocity. Within the functions, the velocity is halved to produce an exploding reflector synthetic. The final three commands create events by the superposition of hyperbolae along linear or piecewise linear structures. By calling these functions repeatedly, the seismic responses from quite complex geometric shapes can be synthesized.

Code Snippet 5.3.1. *This example creates the synthetic zero offset section shown in Figure 5.25.*

```
1    v=2000;dx=10;dt=.004;%basic model parameters
2    x=0:dx:2000;%x axis
3    t=0:dt:2;%t axis
4    seis=zeros(length(t),length(x));%allocate seismic matrix
5    seis=event_hyp(seis,t,x,.4,700,v,1,3);%hyperbolic event
6    seis=event_dip(seis,t,x,[.75 1.23],[700 1500],1);%linear event
7    [w,tw]=ricker(dt,40,.2);%make ricker wavelet
8    seis=sectconv(seis,t,w,tw);%apply wavelet
```

End Code

Code Snippet 5.3.2. *This is similar to Code Snippet 5.3.1 but differs by using* `event_diph` *to create the dipping event. The result is shown in Figure 5.26.*

```
1    v=2000;dx=10;dt=.004;%basic model parameters
2    x=0:dx:2000;%x axis
3    t=0:dt:2;%t axis
4    seis=zeros(length(t),length(x));%allocate seismic matrix
5    seis=event_hyp(seis,t,x,.4,700,v,1,3);%hyperbolic event
6    seis=event_dip(seis,t,x,[.75 1.23],[700 1500],1);%linear event
7    [w,tw]=ricker(dt,40,.2);%make ricker wavelet
8    seis=sectconv(seis,t,w,tw);%apply wavelet
```

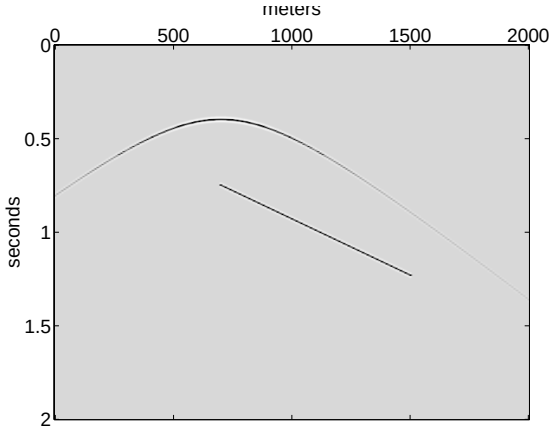

Figure 5.25: The hyperbolic response of a point diffractor and a simple linear event are shown. The display is slightly clipped to make the diffraction more visible. This was created by Code Snippet 5.3.1.

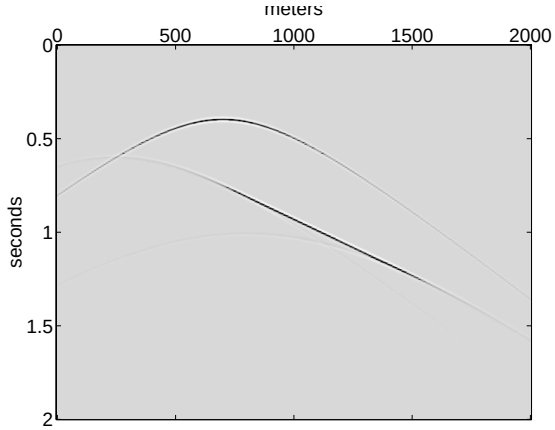

Figure 5.26: Similar to Figure 5.25 except that the linear event was created by the superposition of many hyperbolae. The display is clipped to match that of Figure 5.25. See Code Snippet 5.3.2.

End Code

Code Snippet 5.3.1 illustrates the use of ***event_hyp*** and ***event_dip*** to create the model section shown in Figure 5.25. The first three lines establish the basic model geometry and define the coordinate axes; then, line 4 initializes the seismic matrix to zero. On line 5, ***event_hyp*** is invoked to make the hyperbolic diffraction response shown in the top left of Figure 5.25. The seismic matrix is input to ***event_hyp*** and is replaced by the output. The coordinate vectors (second and third input parameters) are required to defined the geometry. The fourth and fifth input parameters are the (x, t) coordinates of the apex of the diffraction and the sixth parameter is the velocity. The seventh parameter sets the amplitude at the apex of the hyperbola and the eighth is a flag that determines how amplitude decays down the limbs of the hyperbola. There are four possibilities: (1) no amplitude decay, (2) decay given by $a(x) = a(0)t_0/t_x$ (where $a(x)$ is the amplitude at offset x, t_0 is the zero-offset traveltime, and t_x is the traveltime at offset x), (3) decay given by $a(x) = a(0)(t_0/t_x)^{3/2}$, and (4) decay given by $a(x) = a(0)(t_0/t_x)^2$. Line 6 invokes ***event_dip*** to create the linear dipping event seen in the center of Figure 5.25. The fourth input parameter specifies the time of the event at its beginning and end while the fifth parameter gives the corresponding lateral positions. The last parameter gives the event amplitude. Once the events are created, the final two lines make a Ricker wavelet (dominant frequency of 40 Hz) and convolve it with the seismic section.

The linear event shown in Figure 5.25 is physically unrealistic because it lacks diffractions from its endpoints. Real wavefields are never discontinuous. A much more realistic event can by created using ***event_diph***. This function synthesizes a linear event by hyperbolic superposition exactly as illustrated in Figure 5.20. The result is the seismic section shown in Figure 5.26. In order to do so, the input parameters must describe the dipping event in (x, z) rather than (x, t). Code Snippet 5.3.2 shows how this is done and differs from Code Snippet 5.3.1 only on line 6. The fourth input parameter for ***event_diph*** is the velocity while the next four parameters are respectively: the starting and ending x coordinates, the starting z coordinate, and the dip in degrees. The lateral extent of the resulting event is generally much greater than the prescribed coordinates. If the section

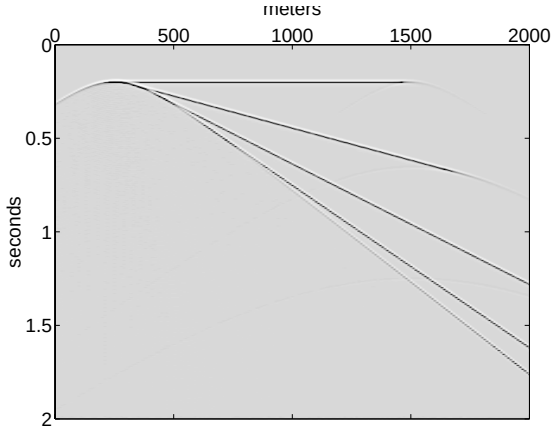

Figure 5.27: The response of five reflectors with dips of 0°, 20°, 40°, 60°, and 80°.

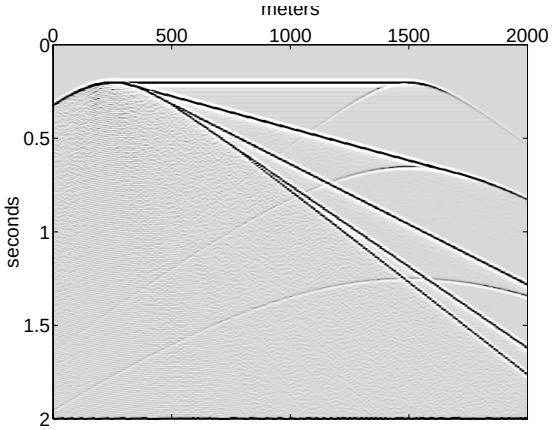

Figure 5.28: The same dataset as Figure 5.28 is shown with a strongly clipped display to reveal the underlying hyperbolae.

is migrated, then the migrated reflector will be found between the specified coordinates.

A more complex seismic section is shown in Figure 5.27. The geologic model is a fan of five dipping reflectors that originate at the point $(x, z) = (250\text{m}, 200\text{m})$ and extend to the right until $x = 1500m$. In Figure 5.28 the same seismic section is shown with a strongly clipped display to reveal the underlying hyperbolae.

Code Snippet 5.3.3. *This code illustrates the use of* `event_pwlinh` *to create a simple model of a reef. The result is shown in Figure 5.29.*

```
1    v=2000;dx=10;dt=.004;%basic model parameters
2    x=0:dx:3000;%x axis
3    t=0:dt:1.5;%t axis
4    zreef=600;hwreef=200;hreef=50;%depth, half-width, and height of reef
5    xcntr=max(x)/2;
6    xpoly=[xcntr-hwreef xcntr-.8*hwreef xcntr+.8*hwreef xcntr+hwreef];
7    zpoly=[zreef zreef-hreef zreef-hreef zreef];
8    seis4=zeros(length(t),length(x));%allocate seismic matrix
9    seis4=event_diph(seis4,t,x,v,0,xcntr-hwreef,zreef,0,.1);%left of reef
10   seis4=event_diph(seis4,t,x,v,xcntr+hwreef,max(x),zreef,0,.1);%right of reef
11   seis4=event_diph(seis4,t,x,v,xcntr-hwreef,xcntr+hwreef,zreef,0,.2);%base of reef
12   seis4=event_pwlinh(seis4,t,x,v,xpoly,zpoly,-.1*ones(size(zpoly)));%reef
13   [w,tw]=ricker(dt,40,.2);%make ricker wavelet
14   seis4=sectconv(seis4,t,w,tw);%apply wavelet
```

End Code

Code Snippet 5.3.3 illustrates the use of these hyperbolic summation tools to create a simple model of a reef. The resulting seismic response is shown in Figures 5.29 and 5.30. The reef is a simple trapezoidal structure, 400 m wide and 50 m high, on top of a flat reflector 600 m below the recording plane. The reflection coefficient of the reef is modelled as -.1 (the acoustic impedance within the reef is assumed to be lower than the surrounding material), +.1 on the base reflector, and +.2 beneath the reef.

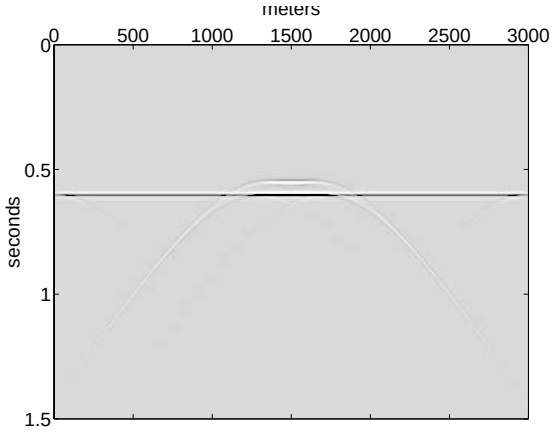

Figure 5.29: The seismic response of a simple reef model. This was created with Code Snippet 5.3.3

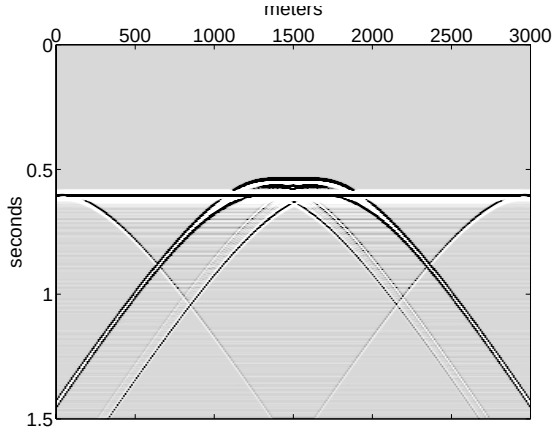

Figure 5.30: The same dataset as Figure 5.29 is shown with a strongly clipped display to reveal the underlying hyperbolae.

Code Snippet 5.3.4. *This code uses* `event_diph2` *to construct the response of a trapezoidal structure. The parameter* `ndelx` *(line 5) controls the spacing of the hyperbolae. Figures 5.31, 5.32, 5.33 and 5.34 correspond to values of* `ndelx` *of 15, 10, 5, and 1 respectively.*

```
1   v=2000;dx=5;dt=.004;%basic model parameters
2   x=0:dx:3000;%x axis
3   t=0:dt:1.5;%t axis
4   xcntr=max(x)/2;
5   ndelx=30;
6   seis5=zeros(length(t),length(x));%allocate seismic matrix
7   seis5=event_diph2(seis5,t,x,v,0,500,1000,ndelx,0,.1);
8   seis5=event_diph2(seis5,t,x,v,500,xcntr-500,1000,ndelx,-45,.1);
9   seis5=event_diph2(seis5,t,x,v,xcntr-500,xcntr+500,500,ndelx,0,.1);
10  seis5=event_diph2(seis5,t,x,v,xcntr+500,max(x)-500,500,ndelx,45,.1);
11  seis5=event_diph2(seis5,t,x,v,max(x)-500,max(x),1000,ndelx,0,.1);
12  [w,tw]=ricker(dt,40,.2);%make ricker wavelet
13  seis5=sectconv(seis5,t,w,tw);%apply wavelet
```

End Code

The function ***event_diph2*** is similar to ***event_diph*** except that it allows control over the spacing of hyperbolae through the input parameter `ndelx`. In ***event_diph*** the hyperbolae spacing is never greater than the grid spacing while in ***event_diph2*** the spacing is `ndelx` times greater than in ***event_diph***. Code Snippet 5.3.4 illustrates the use of this function to create a series of figures illustrating the gradual formation of the seismic response of a trapezoidal structure. The sequence of Figures 5.31, 5.32, 5.33 and 5.34 shows the complete seismic response gradually forming as the density of hyperbolae increases.

Also present in the ***synsections*** toolbox are two functions that make "standard" synthetics called ***makestdsyn*** and ***makestdsynh***. These differ in that one uses hyperbolic superposition for linear events and the other does not. Their use can be illustrated in the script ***makesections*** that can be run as a demonstration.

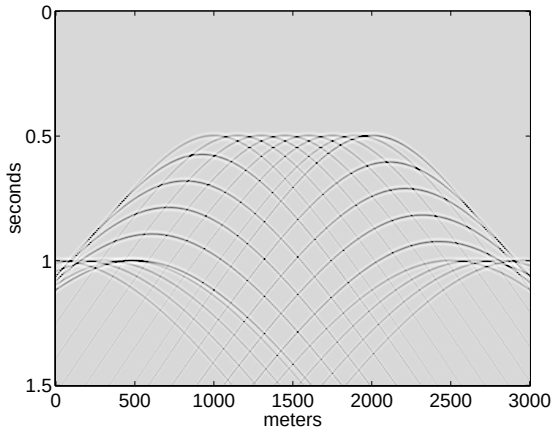

Figure 5.31: A trapezoidal structure is modelled with only a few sparse hyperbolae. Relative to Figure 5.34 only every 15th hyperbola is shown.

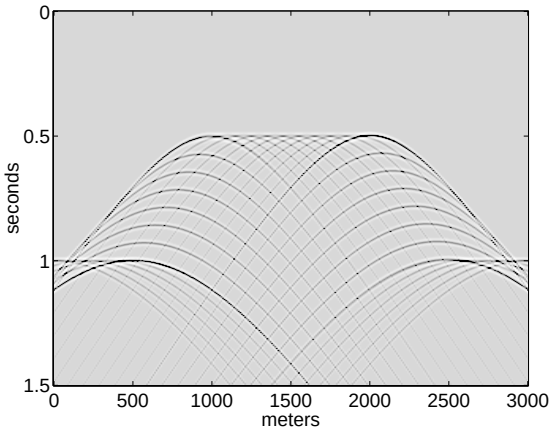

Figure 5.32: Similar to Figure 5.31 except that every 10th hyperbola is shown.

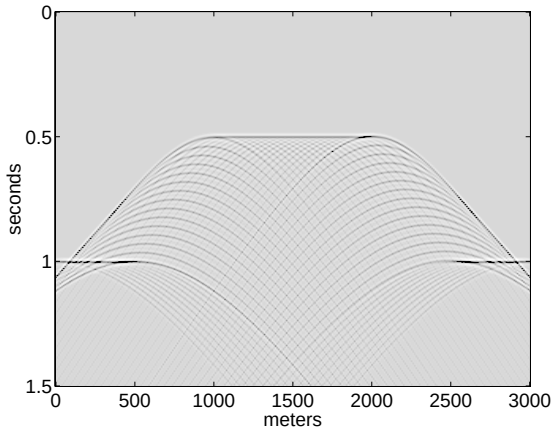

Figure 5.33: Similar to Figure 5.31 except that every 5th hyperbola is shown.

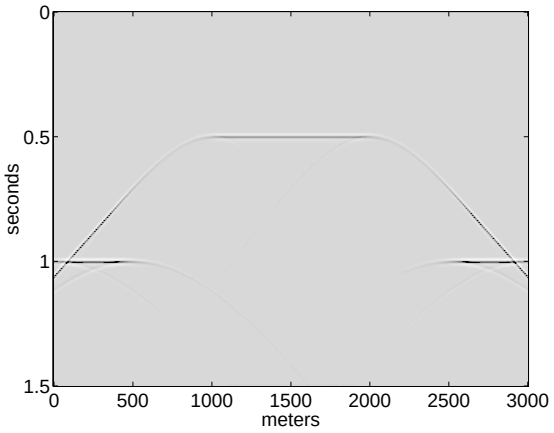

Figure 5.34: A trapezoidal structure is completely modelled by hyperbolic superposition. Compare with Figures 5.31, 5.32, and 5.33.

5.3.2 Finite difference modelling for exploding reflectors

This section builds on section 3.3 and describes those extensions required for exploding reflector modeling. The key facility here is the function ***afd_explode*** that uses an input velocity model to calculate the initial exploding reflector snapshot, $\psi(x, z, t = 0)$, and then time-step it with equation (3.18).

The calculation of $\psi(x, z, t = 0)$ can be done directly from the velocity model if it is assumed to be the normal incidence reflectivity at constant density. Using equation (5.1) and assuming a unit volume, that is

$$\psi(x, z, t = 0) = r_n(x, z) = \frac{1}{2} \left| \vec{\nabla} \log v(x, z) \right|. \tag{5.39}$$

This can be simply calculated using MATLAB's built-in gradient function, `gradient`. Given an input matrix representing $\log v(x, z)$, this function returns two matrices of the same size representing $\partial_x \log v(x, z)$ and $\partial_z \log v(x, z)$. These are calculated using a centered finite difference approximation for interior points and one-sided approximations on the boundaries. Thus, reflectivity is calculated by

```
[rx,rz]=gradient(log(velocity));
r=.5*sqrt(rx.^2+rz.^2);
```

where `velocity` is a matrix representing $v(x, z)$. This calculation is provided by the function ***afd_reflec*** that is invoked by ***afd_explode***.

Code Snippet 5.3.5. *This code uses of* `afd_explode` *to model the exploding reflector response of a small channel. Lines 1-11 build the velocity model and lines 13-25 create a fourth-order exploding reflector seismogram. The results are shown in Figures 5.35, and 5.36.*

```
1    dx=5; %cdp interval
2    xmax=2500;zmax=1000; %maximum line length and maximum depth
3    x=0:dx:xmax; % x coordinate vector
4    z=0:dx:zmax; % z coordinate vector
5    vhigh=4000;vlow=2000; % high and low velocities
6    vrange=vhigh-vlow;
7    %initialize velocity matrix as a constant matrix full of vlow
8    vel=vlow*ones(length(z),length(x));
9    %first layer
10   z1=100;z2=200;v1=vlow+vrange/5;
11   xpoly=[-dx xmax+dx xmax+dx -dx];zpoly=[z1 z1 z2 z2];
12   vel=afd_vmodel(dx,vel,v1,xpoly,zpoly);
13   %second layer
14   z3=271;v2=vlow+2*vrange/5;
15   zpoly=[z2 z2 z3 z3];
16   vel=afd_vmodel(dx,vel,v2,xpoly,zpoly);
17   %third layer
18   z4=398;v3=vlow+pi*vrange/5;
19   zpoly=[z3 z3 z4 z4];
20   vel=afd_vmodel(dx,vel,v3,xpoly,zpoly);
21   %last layer
22   zpoly=[z4 z4 zmax+dx zmax+dx];
23   vel=afd_vmodel(dx,vel,vhigh,xpoly,zpoly);
24   %channel
25   width=20;thk=50;
26   vch=vlow+vrange/6;
27   xpoly=[xmax/2-width/2 xmax/2+width/2 xmax/2+width/2 xmax/2-width/2];
28   zpoly=[z4 z4 z4+thk z4+thk];
29   vel=afd_vmodel(dx,vel,vch,xpoly,zpoly);
```

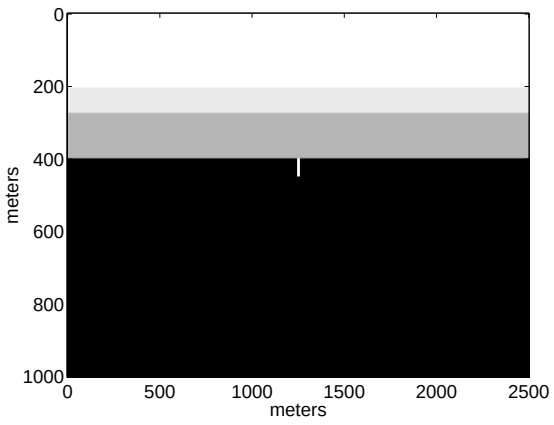

Figure 5.35: The velocity model created by Code Snippet 5.3.5.

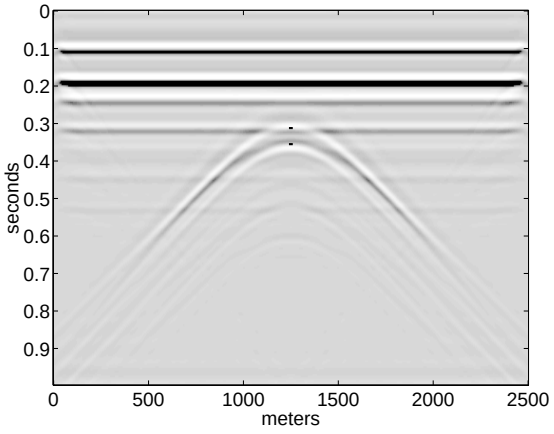

Figure 5.36: The fourth-order exploding reflector seismogram created on line 35 of Code Snippet 5.3.5.

```
30
31   %create the exploading reflector model
32   dt=.004; %temporal sample rate
33   dtstep=.001; %modelling step size
34   tmax=2*zmax/vlow; %maximum time
35   [seisfilt,seis,t,xrec]=afd_explode(dx,dtstep,-dt,tmax, ...
36                   vel,length(x),x,zeros(size(x)),[10 15 40 50],0,2,0);
```

End Code

Code Snippet 5.3.5 illustrates the creation of a model of a small channel beneath a layered medium. The creation of the velocity model uses *afd_vmodel* and was discussed in detail in section 3.3. The channel is defined on lines 25-29 as 20 m wide and 50 m deep. With the 5 m grid (line 1) this means that the channel is 4 samples wide and 10 samples deep. The resulting velocity model is shown in Figure 5.35. Lines 32-36 take this velocity model into *afd_explode* to create an exploding reflector seismogram using the fourth-order Laplacian approximation (equation (3.20)). As with *afd_shotrec*, two temporal sample rates are prescribed. The coarser one (`dt` on line 32) specifies the sample rate of the final seismogram while the finer one (`dtstep` on line 33) is the time step of the modelling. The resulting seismogram, filtered with Ormsby parameters of [10 15 40 50] is shown in Figure 5.36. Two small horizontal lines are superimposed to indicate the position of the channel. The lines are at the time of the top and bottom of the channel (.312 and .355 seconds) and have the same width as the channel. The channel produces an extensive diffraction pattern showing scattered energy over the entire model.

Figure 5.37 shows a second exploding reflector seismogram of the channel model that was calculated with the second-order approximate Laplacian (equation (3.19)). In comparison with the fourth-order seismogram of Figure 5.36, this result is considerably less accurate though it required only one-half of the computation time. Even less accurate is the second-order solution shown in Figure 5.38 that was done with a grid sample size of 10 m rather than the 5m of the previous two figures. This result is almost unrecognizable as the response of the same structure. An obvious conclusion is that the results from finite difference modelling should not be trusted without a careful consideration of the algorithms and parameters.

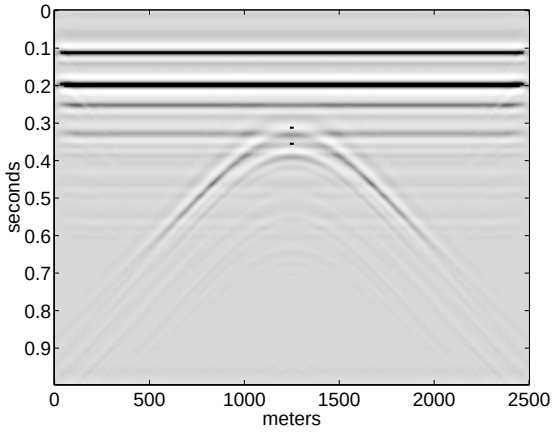

Figure 5.37: Similar to the seismogram of Figure 5.36 except that the Laplacian was approximated with second-order finite differences.

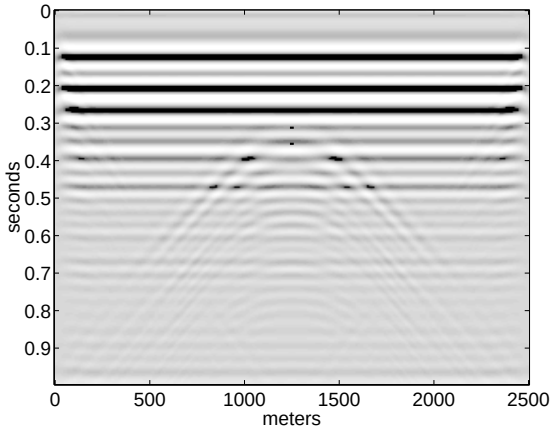

Figure 5.38: Similar to the seismogram of Figure 5.37 except that the spatial sample rate was 10m rather than 5m.

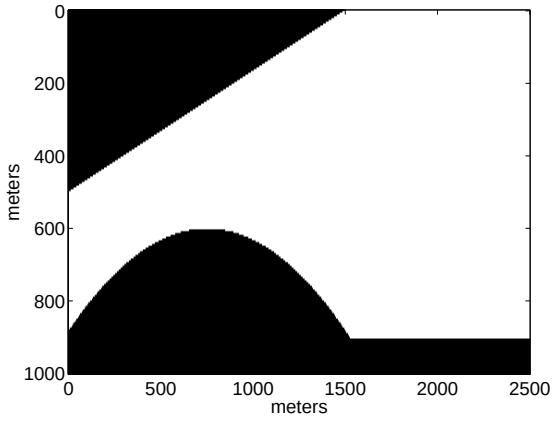

Figure 5.39: An anticline beneath a high velocity wedge. Black is 4000 m/s and white is 2000 m/s.

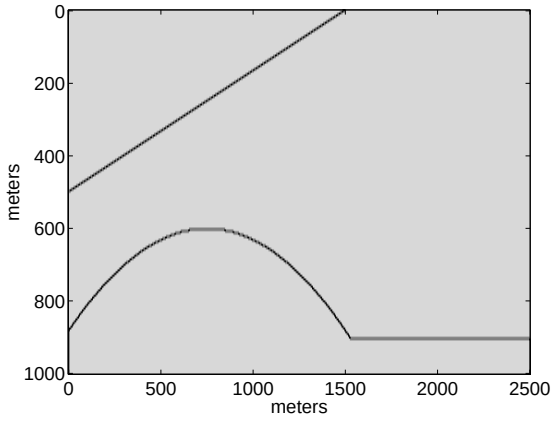

Figure 5.40: The reflectivity, corresponding to Figure 5.39, is shown. This was calculated with *afd_reflect*.

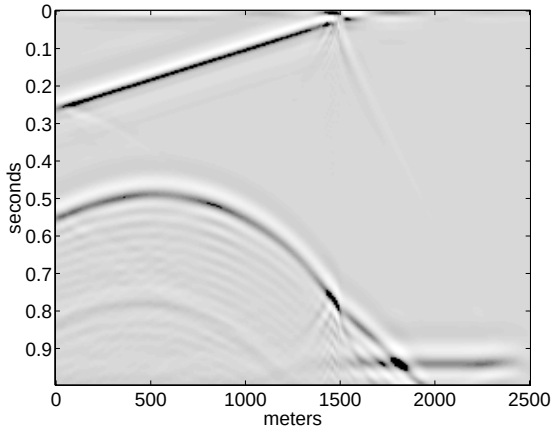

Figure 5.41: The fourth-order exploding reflector seismogram corresponding to the model of Figure 5.39. This model was created using a spatial grid size of 10 m.

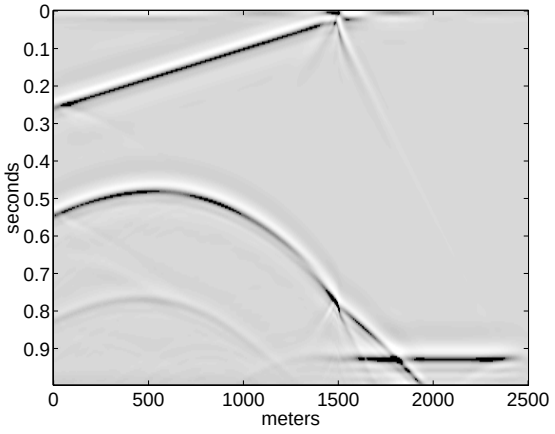

Figure 5.42: Similar to Figure 5.41 except that the spatial grid size was 5 m.

As a slightly more complex example that will be useful in examining depth migration, consider the response of an anticline beneath a high-velocity wedge as shown in Figure 5.39. Code Snippet 5.3.6 creates this velocity model and the fourth-order exploding reflector seismogram shown in Figure 5.41. This result was created with a spatial grid size of 10 m. It might be tempting to trust this result as the proper physical response of the model, but that could lead to erroneous conclusions. For example, it might be concluded that the response of the anticline has a series of reverberations following the primary response. However, before leaping to conclusions, it is prudent to recreate the model with a finer spatial grid size. Figure 5.42 shows the result of re-running Code Snippet 5.3.6 with the spatial grid size set to 5 m. The reverberations have vanished and the amplitude variation of the primary response of the anticline has changed considerably. Once again, it pays to be cautious when working with finite difference synthetics. The only way to be certain is to fully understand the algorithm and how to change the parameters to increase the accuracy. When the accuracy parameters are increased by a factor of two and the response does not change significantly, then it is safe to conclude that the model is showing an accurate physical response. Of course, this can be a tedious process because increasing the accuracy will always increase the computation time.

Code Snippet 5.3.6. *This creates an exploding reflector model of an anticline beneath a high velocity wedge. Lines 1-25 build the velocity model and lines 27-34 create a fourth-order exploding reflector seismogram. The results are shown in Figures 5.39, and 5.41.*

```
1    dx=5; %cdp interval
2    xmax=2500;zmax=1000; %maximum line length and maximum depth
3    xpinch=1500; % wedge pinchout coordinates
4    zwedge=zmax/2; % wedge maximum depth
5    x=0:dx:xmax; % x coordinate vector
6    z=0:dx:zmax; % z coordinate vector
7    vhigh=4000;vlow=2000; % high and low velocities
8    vel=vlow*ones(length(z),length(x));%initialize velocity matrix
9    % define the wedge as a three point polygon
10   dx2=dx/2;
11   xpoly=[-dx2 xpinch -dx2];zpoly=[-1 -1 zwedge];
```

```
% install the wedge in the velocity matrix
vel=afd_vmodel(dx,vel,vhigh,xpoly,zpoly);
% define an anticline beneath the wedge
x0=xpinch/2;z0=zwedge+100; % x and z of the crest of the anticline
a=.0005; % a parameter that determines the steepness of the flanks
za=a*(x-x0).^2+z0; % model the anticline as a parabola
% build a polygon that models the anticline
ind=near(za,zmax+dx);
xpoly=[x(1:ind) 0 ];zpoly=[za(1:ind) za(ind)];
%install the anticline in the velocity model
vel=afd_vmodel(dx,vel,vhigh,xpoly,zpoly);
% bottom layer
xpoly=[0 xmax xmax 0];zpoly=[.9*zmax .9*zmax zmax+dx zmax+dx];
vel=afd_vmodel(dx,vel,vhigh,xpoly,zpoly);

%do a finite-difference model
dt=.004; %temporal sample rate
dtstep=.001;
tmax=2*zmax/vlow; %maximum time
%[w,tw]=wavemin(dt,30,.2); %minimum phase wavelet
%[w,tw]=ricker(dt,70,.2); %ricker wavelet
[seisfilt,seis,t,xrec]=afd_explode(dx,dtstep,dt,tmax, ...
                vel,length(x),x,zeros(size(x)),[5 10 40 50],0,2,0);
```

End Code

5.3.3 Migration and modelling with normal raytracing

Normal raytracing, that is the determination of the raypaths and traveltimes of normal incidence rays, does not directly provide a complete seismic section. The determination of amplitudes would require solution of the geometric spreading equation, (4.73), in addition to the eikonal equation. However, even with this shortcoming, a great deal of useful information can be obtained by using normal raytracing for migration or modelling.

The two key functions here are ***normray*** and ***normraymig*** that do modelling and migration respectively. Migration by normal incidence raytracing was described in section 5.2.2 and normal ray modelling is essentially just the inverse process. Function ***normraymig*** accepts the three parameters defining a pick, $(x_0, t_0, \Delta t/\Delta x)$, (page 123) and then uses ***shootrayvxz*** (discussed in section 4.12.2) to trace that ray down into a velocity model. The ***plotimage*** picking facility (section 1.3.3) can be used to provide these picks for ***normraymig*** to migrate. Function ***normraymig*** returns the abstract ray vector, $\vec{r} = [\vec{x},\ \vec{p}]$ (see equation (4.90)) as an n-by-4 matrix. The first two columns are the (x, z) coordinates while the second two are the horizontal and vertical slownesses. The number of rows is the number of time steps required to trace the ray. Thus the ray vector provides complete specification of the raypath. As a convenience, ***normraymig*** will plot the raypath in a user-specified figure.

The use of ***normraymig***, as described in the previous paragraph, can become quite tedious if there are a great many picks to migrate. Therefore, ***eventraymig*** is provided to automatically migrate all of the picks stored in the global `PICKS`. Function ***eventraymig*** requires only a single input parameter and that is the figure number to draw the raypaths in. However, prior to running either ***normraymig*** or ***eventraymig*** the velocity model must be properly established using ***rayvelmod*** as described in section 4.12.2.

In Figure 5.43, the exploding reflector seismogram of Figure 5.42 is shown with picks on the event from the anticline. These picks were made with the ***plotimage*** picking facility (section 1.3.3) and so are automatically stored as a global variable for ***eventraymig***. Prior to migrating these picks, and after running Code Snippet 5.3.6, the velocity matrices required for raytracing were initialized

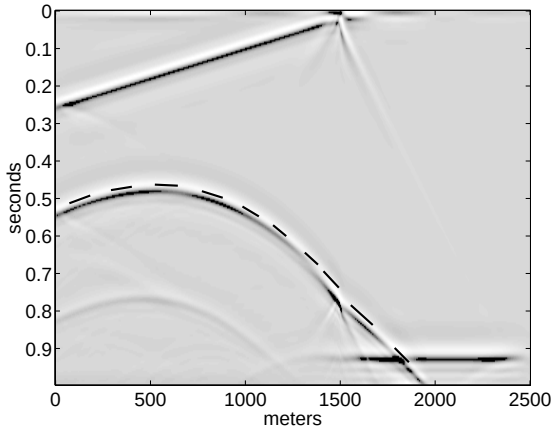

Figure 5.43: The seismogram of Figure 5.42 is shown with picks on the image of the anticline.

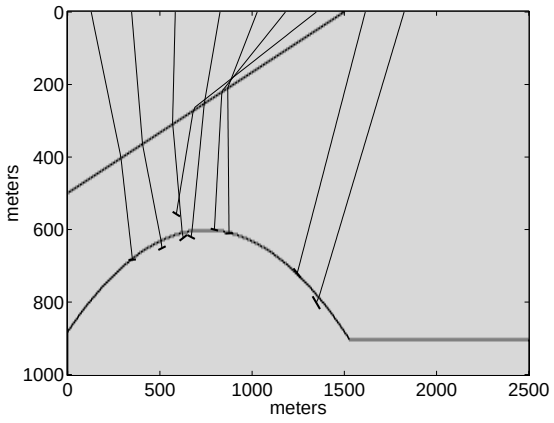

Figure 5.44: The picks of Figure 5.43 are shown migrated on top of the reflectivity section.

with the command `rayvelmod(vel,dx)` where `vel` is the velocity matrix and `dx` is the grid size. Then, the command `eventraymig(figno)`, where `figno` is the MATLAB figure number of Figure 5.39, causes the picks to be migrated and the resulting raypaths displayed in Figure 5.44. At the termination of each raypath, a small line segment, perpendicular to the raypath, is drawn that indicates the implied reflector dip. (These do not appear perpendicular in Figure 5.44 because the (x, z) axes do not have the same scale. The command `axis equal` will display any figure with equal scales on all axes.)

The relative accuracy of the migrations in Figure 5.44 is instructive. The picks have all migrated to positions near the anticline but some have fallen short while others have gone too far. Among the obvious reasons for this are the difficulty in determining which phase (peak, trough, zero crossing, etc.) of the input waveform should be picked and then making a consistent pick at two points. A slight error in either case can result in a very large error in the final position. Since the material beneath the anticline has a high velocity, a pick that arrives at the anticline with a little time left (see section 5.2.2) will continue a significant distance. Thus, small errors in the pick time can make a large error in the result. Also, an error in picking $\Delta t/\Delta x$ will cause the initial trajectory of the ray to be incorrect. Since $\sin\theta_0 = .5v_0\Delta t/\Delta x$, these emergence angle errors are also more significant in high-velocity material.

Migration by normal raytracing can reveal a lot about the nature of a seismic dataset. Also instructive is the complementary process of normal-incidence raytrace modelling. This process is implemented in the function ***normray*** that is logical reverse of ***normraymig***. The latter requires the pick specification of $(x_0, t_0, \Delta t/\Delta x)$ while the former needs the specification of the normal ray: (x_n, z_n, θ_n). Here, (x_n, z_n) are the coordinates of the normal incidence point and θ_n is the structural dip (in degrees) at the normal incidence point. Though logically similar, it is convenient to use separate raytracing engines for these two tasks because they have different criteria for stopping the ray. In migration, the ray is terminated when it has used the available traveltime while in modelling, it is stopped when it encounters the recording surface ($z = 0$). These raytracing engines are ***shootrayvxz*** and ***shootraytosurf*** respectively.

As with the migration tools, it is tedious to invoke ***normray*** at the command line for each pick. Therefore, a convenience function, ***eventraymod*** is provided that automatically models any picks found in the global variable `PICKS`. These picks are expected to have been made on a depth section

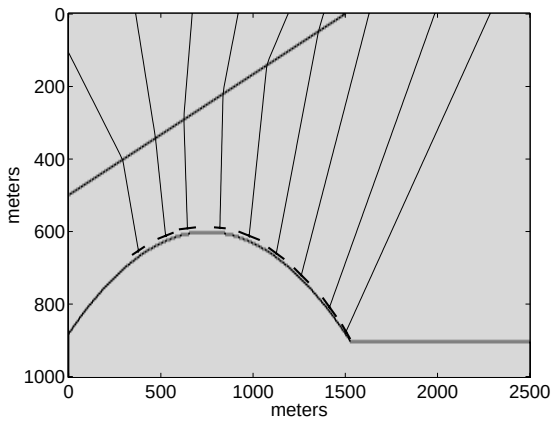

Figure 5.45: The reflectivity section of Figure 5.40 is shown with picks on the anticline and normal rays to the surface.

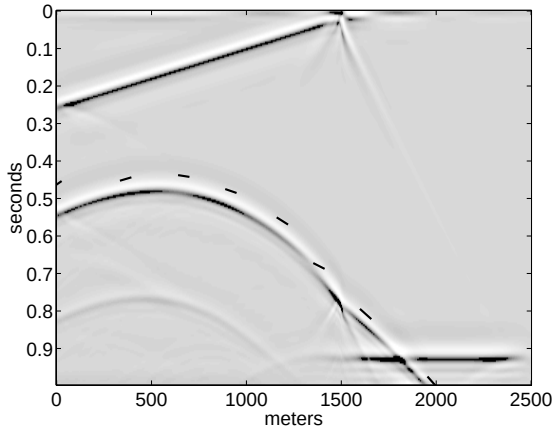

Figure 5.46: The picks of Figure 5.45 are shown modelled on top of the seismic section of Figure 5.42.

though no check is made to ensure this. Figure 5.45 shows the reflectivity section of Figures 5.40 and 5.44 with a series of picks, (x_n, z_n, θ_n), made on the anticline. Also shown are the normal incidence raypaths (drawn by ***normray***) to the surface. Figure 5.46 shows the modelled picks, $(x_0, t_0, \Delta t/\Delta x)$, on top of the seismic section of Figures 5.42 and 5.43.

5.4 Fourier methods

The Fourier methods are developed from an exact solution to the wave equation using Fourier transforms. They provide a high-fidelity migration that illustrates precisely how the migrated spectrum is formed. There are two fundamental approaches and many variations of each. f-k migration (Stolt, 1978) is an exact solution to the migration problem for constant velocity. It is a direct method that is the fastest known migration technique. Phase-shift migration (Gazdag, 1978) is a recursive approach that uses a series of constant velocity extrapolations to build a $v(z)$ migration.

5.4.1 f-k migration

Stolt (1978) showed that the migration problem can be solved by Fourier transform. Here, Stolt's solution will be developed from a formal solution of the wave equation using Fourier transforms. It will be developed in 2D and the direct 3D extension will be stated at the end.

Let $\psi(x, z, t)$ be a scalar wavefield that is a solution to

$$\nabla^2 \psi - \frac{1}{\hat{v}^2} \frac{\partial^2 \psi}{\partial t^2} = 0 \tag{5.40}$$

where $\hat{v}$ is the constant ERM velocity. It is desired to calculate $\psi(x, z, t = 0)$ given knowledge of $\psi(x, z = 0, t)$. The wavefield can be written as an inverse Fourier transform of its (k_x, f) spectrum as

$$\psi(x, z, t) = \int\limits_{\mathbf{V}_\infty} \phi(k_x, z, f) e^{2\pi i (k_x x - f t)} dk_x df \tag{5.41}$$

where cyclical wavenumbers and frequencies are used and the Fourier transform convention uses a + sign in the complex exponential for spatial components and a − sign for temporal components. (The notation for the integration domain is explained in section 5.1.3.) If equation (5.41) is substituted into equation (5.40), the various partial derivatives can be immediately brought inside the integral where they can be readily evaluated. The result is

$$\int\limits_{\mathbf{V}_\infty} \left\{ \frac{\partial^2 \phi(z)}{\partial z^2} + 4\pi^2 \left[\frac{f^2}{\hat{v}^2} - k_x^2 \right] \phi(z) \right\} e^{2\pi i (k_x x - ft)} dk_x df = 0 \tag{5.42}$$

where the (k_x, f) dependence in $\phi(z)$ has been suppressed for simplicity of notation. The derivation of equation (5.42) does not require that $\hat{v}$ be constant; however, the next step does. If $\hat{v}$ is constant[1], then the left-hand-side of equation (5.42) is the inverse Fourier transform of the term in curly brackets. The uniqueness property of Fourier transforms (that there is a unique spectrum for a given function and vice-versa) guarantees that if a function vanishes everywhere in one domain, it must do so in another. Put another way, the zero function has a zero spectrum. Thus, it results that

$$\frac{\partial^2 \phi(z)}{\partial z^2} + 4\pi^2 k_z^2 \phi(z) = 0 \tag{5.43}$$

where the wavenumber k_z is defined by

$$k_z^2 = \frac{f^2}{\hat{v}^2} - k_x^2. \tag{5.44}$$

Equation (5.44) is called the *dispersion relation for scalar waves* though the phrase is somewhat misleading since there is no dispersion in this case.

Equations (5.43) and (5.44) are a complete reformulation of the problem in the (k_x, f) domain. The boundary condition is now $\phi(k_x, z = 0, f)$ which is the Fourier transform, over (x, t), of $\psi(x, z = 0, t)$. Equation (5.43) is a second-order ordinary differential equation for fixed (k_x, f). Either of the functions $e^{\pm 2\pi i k_z z}$ solve it exactly as is easily verified by substitution. Thus the unique, general solution can be written as

$$\phi(k_x, z, f) = A(k_x, f) e^{2\pi i k_z z} + B(k_x, f) e^{-2\pi i k_z z} \tag{5.45}$$

where $A(k_x, f)$ and $B(k_x, f)$ are arbitrary functions of (k_x, f) to be determined from the boundary condition(s). The two terms on the right-hand-side of equation (5.45) have the interpretation of a downgoing wavefield, $A(k_x, f)e^{2\pi i k_z z}$, and an upgoing wavefield, $B(k_x, f)e^{-2\pi i k_z z}$. This can be seen by substituting equation (5.45) into equation (5.41) and determining the direction of motion of the individual Fourier plane waves as described in section 4.9. It should be recalled that z increases downward.

Given only one boundary condition, $\phi(k_x, z = 0, f)$, it is now apparent that this problem cannot be solved unambiguously. It is a fundamental result from the theory of partial differential equations that *Cauchy* boundary conditions (e.g. knowledge of both ψ and $\partial_z \psi$) are required on an open surface in order for the wave equation to have a unique solution. Since this is not the case here, the migration problem is said to be ill-posed. If both conditions were available, A and B could be found as the solutions to

$$\phi(z = 0) \equiv \phi_0 = A + B \tag{5.46}$$

[1]Actually $\hat{v}(z)$ could be tolerated here. The necessary condition is that $\hat{v}$ must not depend upon x or t.

and

$$\frac{\partial \phi}{\partial z}(z=0) \equiv \phi_{z0} = 2\pi i k_z A - 2\pi i k_z B. \tag{5.47}$$

When faced with the need to proceed to a solution despite the fact that the stated problem does not have a unique solution, a common approach is to assume some limited model that removes the ambiguity. The customary assumption of *one-way waves* achieves this end. That is, $\psi(x,z,t)$ is considered to contain only upgoing waves. This allows the solution

$$A(k_x,f)=0 \quad \text{and} \quad B(k_x,f)=\phi_0(k_x,f) \equiv \phi(k_x,z=0,f). \tag{5.48}$$

Then, the ERM wavefield can be expressed as the inverse Fourier transform

$$\psi(x,z,t) = \int\limits_{\mathbf{V}_\infty} \phi_0(k_x,f) e^{2\pi i(k_x x - k_z z - ft)} dk_x df. \tag{5.49}$$

The migrated solution is

$$\psi(x,z,t=0) = \int\limits_{\mathbf{V}_\infty} \phi_0(k_x,f) e^{2\pi i(k_x x - k_z z)} dk_x df. \tag{5.50}$$

Equation (5.50) gives a migrated depth section as a double integration of $\phi_0(k_x,f)$ over f and k_x. Though formally complete, it has the disadvantage that only one of the integrations, that over k_x, is a Fourier transform that can be done rapidly as a numerical FFT. The f integration is not a Fourier transform because the Fourier kernel $e^{-2\pi f t}$ was lost when the imaging condition (setting $t=0$) was invoked. Inspection of equation (5.50) shows that another complex exponential $e^{-2\pi i k_z z}$ is available. Stolt (1978) suggested a change of variables from (k_x,f) to (k_x,k_z) to obtain a result in which both integrations are Fourier transforms. The change of variables is actually defined by equation (5.44) that can be solved for f to give

$$f = \hat{v}\sqrt{k_x^2 + k_z^2}. \tag{5.51}$$

Performing the change of variables from f to k_z according to the rules of calculus transforms equation (5.50) into

$$\psi(x,z,t=0) = \int\limits_{\mathbf{V}_\infty} \phi_m(k_x,k_z) e^{2\pi i(k_x x - k_z z)} dk_x dk_z \tag{5.52}$$

where

$$\phi_m(k_x,k_z) \equiv \frac{\partial f(k_z)}{\partial k_z} \phi_0(k_x, f(k_z)) = \frac{\hat{v} k_z}{\sqrt{k_x^2 + k_z^2}} \phi_0(k_x, f(k_z)) \tag{5.53}$$

Equation (5.52) is Stolt's expression for the migrated section and forms the basis for the f-k migration algorithm. The change of variables has recast the algorithm into one that can be accomplished with FFT's doing all of the integrations. Equation (5.53) results from the change of variables and is a prescription for the construction of the (k_x,k_z) spectrum of the migrated section from the (k_x,f) spectrum of the ERM seismogram.

Many of the important properties of post-stack migration can be discerned from Stolt's result. First, notice that k_z defined though

$$k_z = \sqrt{\frac{f^2}{\hat{v}^2} - k_x^2} \tag{5.54}$$

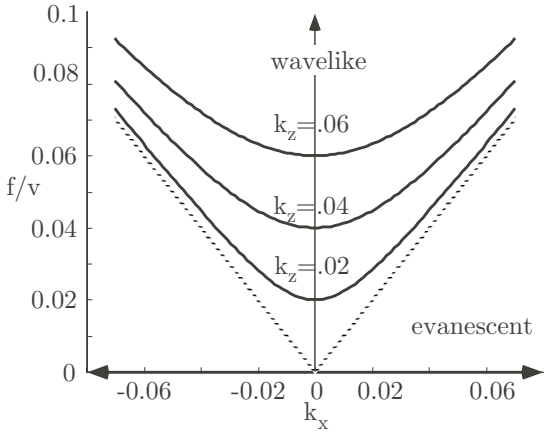

Figure 5.47: The space $(f/v, k_x)$ is shown contoured with values of k_z from equation (5.54). The dashed lines are the boundary between the wavelike and evanescent regions.

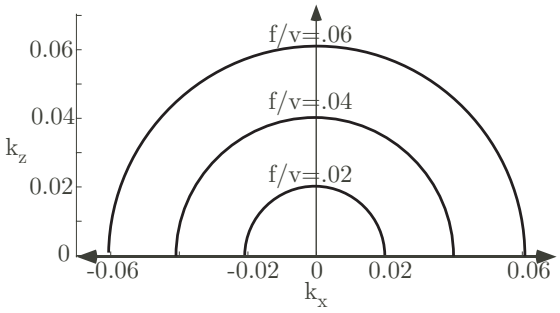

Figure 5.48: The space (k_x, k_z) is shown with contours of f/v as given by equation (5.51).

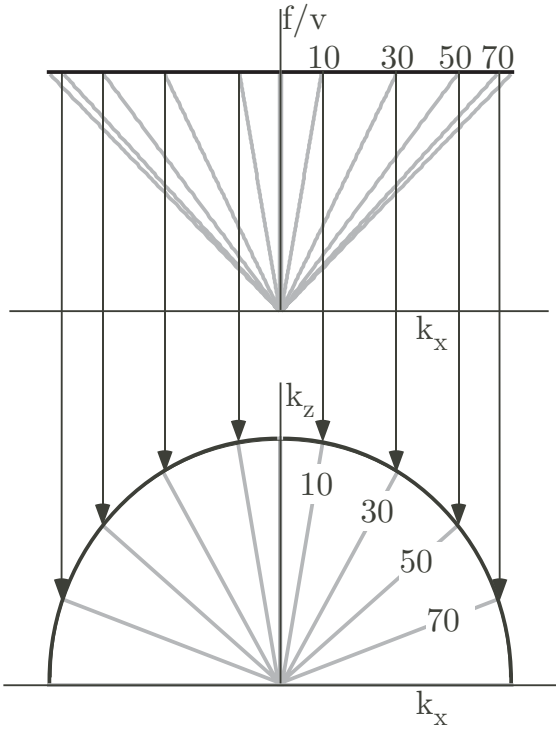

Figure 5.49: The mapping from (k_x, f) (top) to (k_x, k_z) (bottom) is shown. A line of constant f is mapped, at constant k_x, to a semi-circle in (k_x, k_z). The compressed apparent dip spectrum of $(f/v, k_x)$ space unfolds into the uniform fan in (k_x, k_z) space. The numbers on each space indicate dips in degrees.

is real-valued when $|f/k_x| \geq \hat{v}$ and is otherwise imaginary. Only for real k_z will equation (5.45) correspond to traveling waves in the positive and negative z directions. For complex k_z $e^{\pm 2\pi i k_z z}$ becomes a real exponential that either grows or decays; however, on physical grounds, growing exponentials must be excluded. Given a value for $\hat{v}$ this dual behavior of k_z divides (k_x, f) space into two regions as was discussed from another perspective in section 4.11.1. The region where $|f/k_x| \geq \hat{v}$ is called the *wavelike* or *body wave* region and where $|f/k_x| < \hat{v}$ it is called the *evanescent* region. Equation (5.54) is sometimes called a dispersion relation for one-way waves since the choice of the plus sign in front of the square root generates upward traveling waves in $e^{\pm 2\pi i k_z}$ while the minus sign generates downward traveling waves.

The geometric relationships between the spaces of $(f/v, k_x)$ and (k_x, k_z) are shown from two different perspectives in Figures 5.47 and 5.48. In $(f/v, k_x)$ space, the lines of constant k_z are hyperbolae that are asymptotic to the dashed boundary between the wavelike and evanescent regions. In (k_x, k_z) space, the curves of constant f/v are semi-circles. At $k_x = 0$, $k_z = f/v$ so these hyperbolae and semi-circles intersect when the plots are superimposed.

The spectral mapping required in equation (5.53) is shown in Figure 5.49. The mapping takes a constant f slice of (k_x, f) space to a semi-circle in (k_x, k_z) space. Each point on the f slice maps at constant k_x which is directly down in the figure. It is completely equivalent to view the mapping as a flattening of the k_z hyperbolae of Figure 5.47. In this sense, it is conceptually similar to the NMO removal in the time domain though here the samples being mapped are complex valued. That the spectral mapping happens at constant k_x is a mathematical consequence of the fact that k_x is held constant while the f integral in equation (5.50) is evaluated. Conceptually, it can also be viewed as a consequence of the fact that the ERM seismogram and the migrated section must agree at $z = 0$ and $t = 0$.

On a numerical dataset, this spectral mapping is the major complexity of the Stolt algorithm. Generally, it requires an interpolation in the (k_x, f) domain since the spectral values that map to grid nodes in (k_x, k_z) space cannot be expected to come from grid nodes in (k_x, f) space. In order to achieve significant computation speed that is considered the strength of the Stolt algorithm, it turns out that the interpolation must always be approximate. This causes *artifacts* in the final result. This issue will be discussed in more detail in section 5.4.2.

The creation of the migrated spectrum also requires that the spectrum be scaled by $\hat{v}k_z/\sqrt{k_x^2 + k_z^2}$ as it is mapped (Equation (5.53)). In the constant velocity medium of this theory, $\sin\delta = \hat{v}k_x/f$ (δ is the scattering angle) from which it follows that $\cos\delta = \hat{v}k_z/f = k_z/\sqrt{k_x^2 + k_z^2}$. Thus the scaling factor is proportional to $\cos\delta$ and therefore ranges from unity to zero as δ goes from zero to 90°. This scaling factor compensates for the "spectral compression" that is a theoretical expectation of the ERM seismogram. Recall the migrator's formulae (equation (5.35)) that relates apparent angles in $(f/v, k_x)$ space to real angles in (k_x, k_z) space. If there is uniform power at all angles in (k_x, k_z) space, then the migrator's formula predicts a spectral compression in $(f/v, k_x)$ (with consequent increasing power) near 45° of apparent dip. As shown in Figure 5.49 it is as though (k_x, k_z) space is an oriental fan that has been folded to create $(f/v, k_x)$ space. Migration must then unfold this fan. *f-k* migration is called a *steep-dip* method because it works correctly for all scattering angles from 0° to 90°.

The *f-k* migration algorithm just described is limited to constant velocity though it is exact in this case. Its use of Fourier transforms for all of the numerical integrations means that it is computationally very efficient. Though it has been used for many years in practical applications its restriction to constant velocity is often unacceptable so that it has gradually being replaced by more flexible, though usually slower, methods. Today, one major virtue still remains and that is the conceptual understanding it provides. The description of the construction of the migrated spectrum will provide the basis for a realistic theory of seismic resolution in section 5.7.

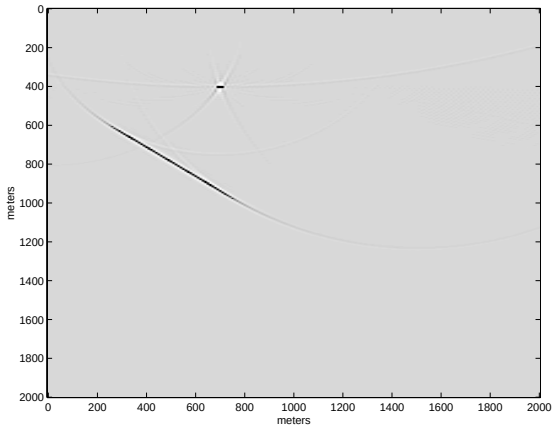

Figure 5.50: The result of the f-k migration of the data of Figure 5.25.

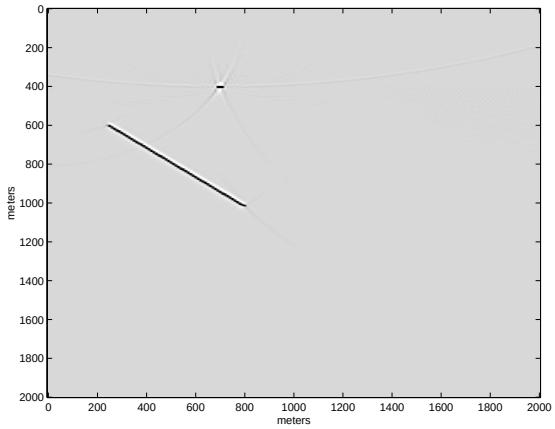

Figure 5.51: The result of the f-k migration of the data of Figure 5.26.

5.4.2 A MATLAB implementation of f-k migration

The f-k migration algorithm just described is a three step process

- Forward (k_x, f) transform. This is done on the unmigrated data after all pre-conditioning such as spectral whitening, bandpass filtering, and noise reduction.
- Spectral mapping. The spectral mapping and scaling described by equation (5.53) requires a careful interpolation process.
- Inverse (k_x, f) transform. This must be the exact inverse of the transform done in the first step.

This process is formalized in the MATLAB function ***fkmig***. This function has the external interface: `[seismig,tmig,xmig] = fkmig(seis,t,x,v,params)`. Here the first four input variables are simply the seismic matrix, its time coordinate vector, its x coordinate vector, and a scalar giving the velocity of migration. The final input variable is a vector of parameters that control various aspects of the migration. This vector, `params` has length thirteen and affords control over the spatial and temporal zero pads, the maximum dip to migrate, the maximum frequency to migrate, the type of spectral interpolation, and the kind of progress messages that are written to the screen. Consult the online help for ***fkmig*** for a complete description. Usually, the default actions for all of these behaviors are acceptable and `params` can be omitted. Occasionally, it will be desired to program one or more elements of `params`. This can be done while still defaulting the others by first creating a vector of thirteen NaN's (e.g. `params=nan*1:13;`) and then setting a few elements of `params` to specific values.

As a first example, consider the migration of the synthetic sections shown in Figures 5.25 and 5.26. This is very simply accomplished with the code in Code Snippet 5.4.1 with the results in Figures 5.50 and 5.51. These figures are displayed with slight clipping to make visible their more subtle details. In Code Snippet 5.4.1, the variable `seis` refers to the data of Figure 5.25 while `seis2` refers to Figure 5.26. Figure 5.25 contains the image of a dipping reflecting segment without diffractions while Figure 5.26 contains a similar image except that diffractions are present. The resulting migrations make

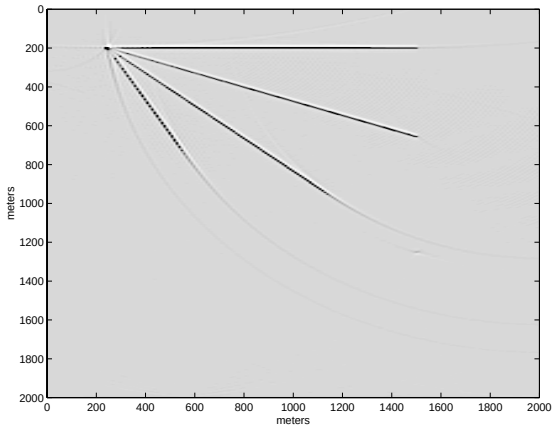

Figure 5.52: The result of the f-k migration of the data of Figure 5.27.

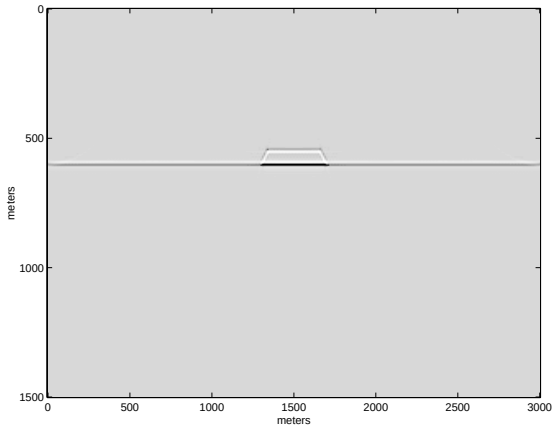

Figure 5.53: The result of the f-k migration of the data of Figure 5.29.

clear the advantage (indeed, the necessity) of modelling with diffractions. Only when diffractions are included are the edges of the reflecting segment imaged with crisp definition.

Code Snippet 5.4.1. *This code uses* `fkmig` *to migrate the sections shown in Figures 5.25 and 5.26. The results are shown in Figures 5.50, and 5.51.*

```
1    seismig=fkmig(seis,t,x,v);
2    plotimage(seismig,t*v/2,x);
3    xlabel('meters');ylabel('meters');
4    seismig2=fkmig(seis2,t,x,v);
5    plotimage(seismig2,t*v/2,x);
6    xlabel('meters');ylabel('meters');
```

End Code

As further examples, the f-k migrations of the data of Figures 5.27, 5.29, 5.31, and 5.34 can be migrated in similar fashion. The results are shown in Figure 5.52, 5.53, 5.54, and 5.55.

It is apparent that ***fkmig*** creates high quality migrations of constant-velocity synthetics, but it is instructive to examine the code to see precisely how this is done. Code Snippet 5.4.2 contains an excerpt from the source code of ***fkmig*** that illustrates the steps of basic geophysical importance. The forward f-k transform is accomplished on line 2 by calling ***fktran***. Prior to this, the seismic matrix, `seis`, has had zero pads attached in both x (column) and t (row). The variables `tnew` and `xnew` are time coordinate and x coordinate vectors that describe the padded seismic matrix. Similarly, `nsampnew` and `ntrnew` are the number of samples-per-trace and the number of traces after padding. Function ***fktran*** is a geophysical wrapper around MATLAB's built-in two-dimensional FFT `fft2`. This means that, in addition to calling `fft2` to compute the f-k spectrum, `fkspec`, of `seis`, it does some useful things like calculating the frequency and wavenumber coordinate vectors `f` and `kx`. Also, because the input matrix is real valued, it only returns the non-negative temporal frequencies. This is possible because the negative frequencies can be deduced from the positive ones by a symmetry argument. Technically, if $\phi(k_x, f)$ is the f-k transform of the real-valued wavefield $\psi(x,t)$ then it can be shown that $\phi(k_x, f)$ has the symmetry $\phi(k_x, f) = \bar{\phi}(-k_x, -f)$, where the overbar indicates the complex conjugate. Both positive and negative wavenumbers are required since, after the temporal Fourier transform, the matrix is no longer real valued. Working only with

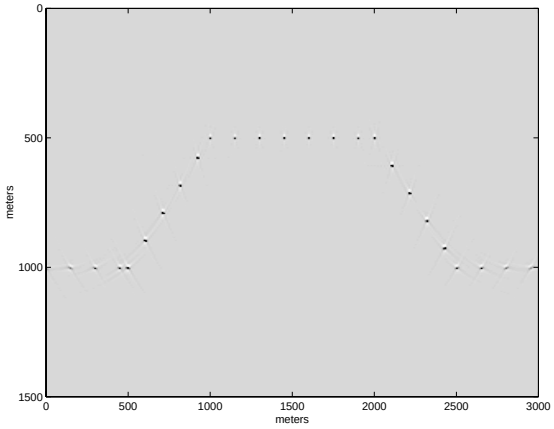

Figure 5.54: The result of the f-k migration of the data of Figure 5.31.

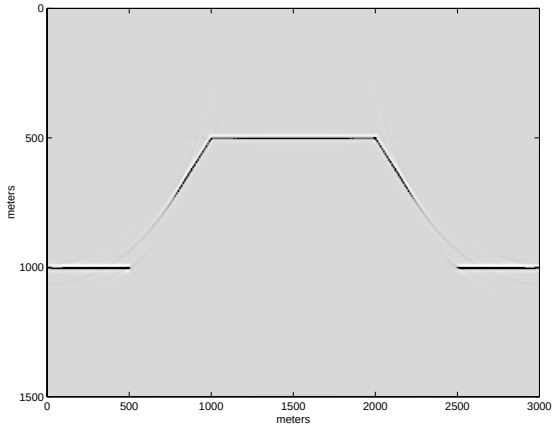

Figure 5.55: The result of the f-k migration of the data of Figure 5.34.

the non-negative temporal frequencies is more efficient because less memory and fewer calculations are required. Also, it requires more care to formulate an algorithm correctly to process both positive and negative frequencies. If the processed spectrum does not have the correct conjugate symmetry then its inverse transform will result in a complex-valued seismic matrix. It is easier to process only the non-negative frequencies and calculate the negative ones as needed from the symmetry condition. The function ***fktran*** has a companion inverse ***ifktran*** (invoked on line 34) that creates the negative temporal frequencies from the non-negative ones and then calls `fft2`.

Continuing with Code Snippet 5.4.2, line 3 calculates the exploding reflector velocity that is required in the theory. The major computation loop is over k_x (the columns of `fkspec`) and extends from line 9 to line 32. Each iteration through the loop converts one column of `fkspec`, representing ϕ_0 of equation (5.53), into a column vector representing ϕ_m. Prior to the loop, lines 5-6 compute the vertical coordinate vector, `kz`, for the matrix representing ϕ_m (`df` is the frequency sample rate). On line 7, k_z^2 is pre-computed as this is needed in every loop. The first action in the loop (line 11) is to apply frequency and dip *masks* to the relevant column of `fkspec` and the result is stored in the temporary vector `tmp`. The calculation of these masks is not shown in this example. They are simply vectors of the same size as a column of `fkspec` whose entries range between zero and one. The frequency mask, `fmask`, is pre-computed outside the loop but the dip mask, `dipmask` must be recalculated with every iteration. These masks are designed to attenuate the frequencies and dips that are not of interest. For example, it is customary to migrate only frequencies below a certain maximum, f_{max}. Thus a simple `fmask` could could be unity for $f \leq f_{max}$ and zero for $f > f_{max}$. However, from a signal processing perspective, such an abrupt cutoff is not desirable so a better `fmask` might be

$$f_{mask} = \begin{cases} 1, f \leq f_{max} - f_{wid} \\ \frac{1}{2} + \frac{1}{2}\cos\left(\pi \frac{f - f_{max} + f_{wid}}{f_{wid}}\right), f_{max} - f_{wid} < f \leq f_{max} \\ 0, f > f_{max} \end{cases} . \tag{5.55}$$

This defines a *raised-cosine ramp* of width f_{wid} from unity at $f = f_{max} - f_{wid}$ to zero at $f = f_{max}$. The dip mask must be recomputed at every loop iteration because the frequency corresponding to

a given dip changes as k_x changes according to $f = k_x \hat{v}/\sin\theta$. Like the frequency mask, the dip mask should use a raised-cosine ramp from zero at θ_{max} to unity at $\theta_{max} - \theta_{wid}$. Thus `dipmask` attenuates low frequencies while `fmask` attenuates high frequencies.

Next, on line 13 in Code Snippet 5.4.2, the frequencies that map to each output k_z called `fmap`, are calculated via equation (5.51). Since the spectral mapping generally lowers frequencies (see Figure 5.49) many of these frequencies will, in general, be greater than the maximum desired frequency to migrate. Thus, on line 14, MATLAB's powerful `find` command is used to identify the entries in the vector `fmap` that are lower than the maximum frequency to migrate, `fmapmig`. (`fmapmig` corresponds to f_{max} in equation (5.55).) At this stage, the vector `ind` is a vector of indices into `fmap` that points to those frequencies that will be migrated.

On line 16 the current column of `fkspec` is set to zero in preparation for the actual migration that happens in the if-block from line 17 to line 28. Lines 18-25 compute the cosine scale factor that occurs in equation (5.53) with a special case for $f = 0$ to avoid division by zero. Line 27 does the actual spectral mapping and applies the cosine scale factor. The spectral mapping is accomplished by a *sinc-function interpolation*, that is optimized for complex-valued spectra, by the function ***csinci***.

On lines 29-31, a progress message is printed. Though not strictly necessary, this is considered *good form* because there are few things more disconcerting than waiting for a computer to finish a long calculation without any indication that progress is occurring. A message is printed only after every `kpflag` wavenumbers have been processed. This is controllable through the input `params` vector.

Finally, after the completion of the loop on line 34, the migrated spectrum is inverse transformed. Generally, the zero pads are then removed though this is not shown.

Code Snippet 5.4.2. *This is an excerpt from the code of* `fkmig` *that shows the steps of major importance. The actual* `fkmig` *source code contains 319 lines that handle the many non-technical details required for a useful code.*

```
1    %forward fk transform
2    [fkspec,f,kx] = fktran(seis,tnew,xnew,nsampnew,ntrnew,0,0);
3    ve = v/2; %exploding reflector velocity
4    %compute kz
5    dkz= df/ve;
6    kz = ((0:length(f)-1)*dkz)';
7    kz2=kz.^2;
8    %now loop over wavenumbers
9    for j=1:length(kx)
10   % apply masks
11       tmp=fkspec(:,j).*fmask.*dipmask;
12   %compute f's which map to kz
13       fmap = ve*sqrt(kx(j)^2 + kz2);
14       ind=find(fmap<=fmaxmig);
15   %now map samples by interpolation
16       fkspec(:,j) = zeros(length(f),1); %initialize output spectrum to zero
17       if( ~isempty(ind) )
18           %compute cosine scale factor
19           if( fmap(ind(1))==0)
20               scl=ones(size(ind));
21               li=length(ind);
22               scl(2:li)=ve*kz(ind(2:li))./fmap(ind(2:li));
23           else
24               scl=ve*kz(ind)./fmap(ind);
25           end
26           %complex sinc interpolation
27           fkspec(ind,j) = scl.*csinci(tmp,f,fmap(ind),[lsinc,ntable]);
```

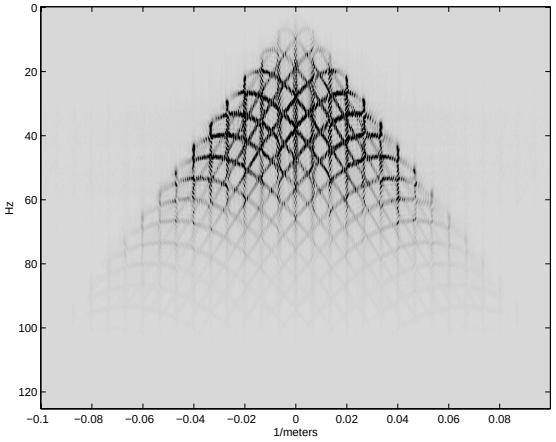

Figure 5.56: The f-k spectrum of the data of Figure 5.31.

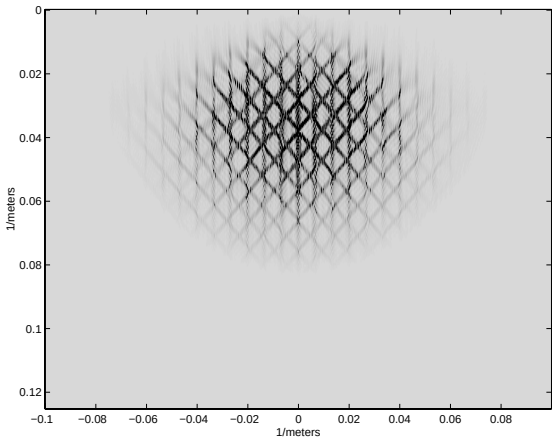

Figure 5.57: The (k_x, k_z) spectrum of the data of Figure 5.54.

```
28        end
29        if( floor(j/kpflag)*kpflag == j)
30            disp(['finished wavenumber ' int2str(j)]);
31        end
32    end
33    %inverse transform
34    [seismig,tmig,xmig]=ifktran(fkspec,f,kx);
```

End Code

As a last example, consider the computation of the discrete f-k spectra of one of the preceding examples before and after migration. This should produce a graphical confirmation of the mapping of Figure 5.49. This is very simply done using ***fktran***. Specifically, the case of the synthetic of Figure 5.31 and its migration in Figure 5.54 is shown. If `seis` is the unmigrated seismic matrix, then the command `[fkspec,f,kx]=fktran(seis,t,x)` computes the complex-valued f-k spectrum and `plotimage(abs(fkspec),f,kx)` produces the result shown in Figure 5.56. In a similar manner, the (k_x, k_z) spectrum after migration can be computed and is shown in Figure 5.57. Comparison of these figures shows that the spectral mapping has been performed as described.

5.4.3 f-k wavefield extrapolation

Stolt (1978) provided an approximate technique to adapt f-k migration to $v(z)$. This method used a pre-migration step called the *Stolt stretch* that was followed by an f-k migration. The idea was to perform a one-dimensional time-to-depth conversion with $v(z)$ and then convert back to a pseudo time with a constant *reference velocity*. The f-k migration is then performed with this reference velocity. (Stolt actually recommended the time-to-depth conversion be done with a special velocity function derived from $v(z)$ called the Stolt velocity.) This method is now known to progressively lose accuracy with increasing dip and has lost favor.

A technique that can handle all scattering angles in $v(z)$ is the phase-shift method of Gazdag (1978). Unlike the direct f-k migration, phase shift is a recursive algorithm that treats $v(z)$ as a system of constant velocity layers. In the limit as the layer thickness shrinks to infinitesimal, any $v(z)$ variation can be modelled. The method can be derived starting from equation (5.49).

Considering the first velocity layer only, this result is valid for any depth within that layer provided that $\hat{v}$ is replaced by the first layer velocity, $\hat{v}_1$. If the thickness of the first layer is Δz_1, then the ERM wavefield just above the interface between layers 1 and 2 can be written as

$$\psi(x, z = \Delta z_1, t) = \int\limits_{\mathbf{V}_\infty} \phi_0(k_x, f) e^{2\pi i(k_x x - k_{z1}\Delta z_1 - ft)} dk_x df. \tag{5.56}$$

where

$$k_{z1} = \sqrt{\frac{f^2}{\hat{v}_1^2} - k_x^2}. \tag{5.57}$$

Equation (5.56) is an expression for *downward continuation* or *extrapolation* of the ERM wavefield to the depth Δz_1. The extrapolated wavefield is distinguished from the surface recorded wavefield by the presence of the term $e^{2\pi i k_{z1}\Delta z_1}$ under the integral sign that is a specific form of the Fourier *extrapolation operator* , $e^{2\pi i k_z \Delta z}$. Any extrapolated wavefield is a temporal seismogram more akin to a surface recording than to a migrated depth section. In the phase-shift method, as with any recursive technique, the migrated depth section is built little-by-little from each extrapolated seismogram. The extrapolated wavefield is a simulation of what would have been recorded had the receivers actually been at $z = \Delta z$ rather than $z = 0$. Since any extrapolated section intersects the depth section at $(z = \Delta z, t = 0)$ each extrapolation can contribute one depth sample to the migration (see Figure 5.24). This process of evaluating the extrapolated section at t=0 was discussed in section 5.2.6 as the post-stack imaging condition.

For the wavelike portion of (k_x, f) space, the extrapolation operator has unit amplitude and a phase of $2\pi k_z \Delta z$. For evanescent spectral components, it is a real exponential $e^{\pm 2\pi |k_z| \Delta z}$. Forward wavefield propagation must obey physical law and propagate evanescent spectral components using the minus sign in the exponent, e.g. $e^{-2\pi |k_z| \Delta z}$. Therefore, inverse wavefield extrapolation, as is done for migration, should use $e^{+2\pi |k_z| \Delta z}$. However, this inversion of evanescent spectral components is a practical impossibility because they have decayed far below the noise level in forward propagation. The practical approach is to use $e^{-2\pi |k_z| \Delta z}$ for the evanescent spectral components for both forward and inverse extrapolation. Even more practical is to simply zero the evanescent spectral components on inverse extrapolation. In all that follows, it is assumed that $e^{2\pi i k_z \Delta z}$has one of these two practical extensions implemented for evanescent spectral components.

The wavefield extrapolation expression (equation (5.56)) is more simply written in the Fourier domain to suppress the integration that performs the inverse Fourier transform:

$$\phi(k_x, z = \Delta z_1, f) = \phi_0(k_x, f) e^{2\pi i k_{z1} \Delta z_1}. \tag{5.58}$$

Now consider a further extrapolation to estimate the wavefield at the bottom of layer 2 ($z = \Delta z_1 + \Delta z_2$). This can be written as

$$\phi(k_x, z = \Delta z_1 + \Delta z_2, f) = \phi(k_x, z = \Delta z_1, f) T(k_x, f) e^{2\pi i k_{z2} \Delta z_2} \tag{5.59}$$

where $T(k_x, f)$ is a correction factor for the transmission loss endured by the upgoing wave as it crossed from layer 2 into layer 1. If transmission loss correction is to be incorporated, then the data must not have had such amplitude corrections applied already. This is extremely unlikely because seismic data is generally treated with a statistical amplitude balance that will compensate for transmission losses. Also, correcting for transmission losses at each extrapolation step can be numerically unstable because the correction factors are generally greater than unity. Consequently, it is customary to set $t(k_x, f) = 1$. With this and incorporating equation (5.58), equation (5.59)

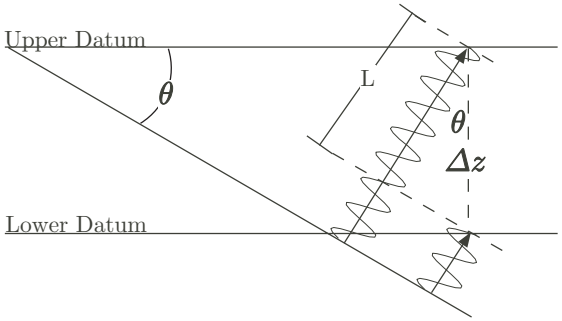

Figure 5.58: A geometric interpretation of the extrapolation phase shift. A dipping reflector emits a monochromatic signal that is received at two different datums. The extrapolation phase shift is the phase difference between this signal as it arrives at a specific (x_0, z_0) location on the upper datum and the signal as it arrives at the lower datum at the same horizontal coordinate, i.e. at $(x_0, z_0 + \Delta z)$.

becomes

$$\phi(k_x, z = \Delta z_1 + \Delta z_2, f) = \phi_0(k_x, f) e^{2\pi i \sum_{m=1}^{2} k_{zm} \Delta z_m} \tag{5.60}$$

which has the immediate generalization to n layers

$$\phi(k_x, z = \sum_{m=1}^{n} \Delta z_m, f) = \phi_0(k_x, f) e^{2\pi i \sum_{m=1}^{n} k_{zm} \Delta z_m}. \tag{5.61}$$

In the limit as $\Delta z_m \to dz$, the summation in the extrapolator phase becomes an integral with the result

$$\phi(k_x, z, f) = \phi_0(k_x, f) e^{2\pi i \int_0^z k_z(z')dz'}. \tag{5.62}$$

where $k_z(z) = \sqrt{f^2/\hat{v}(z)^2 - k_x^2}$. This result is known as a first-order WKBJ[2] solution and can be derived as an approximate solution to the scalar wave equation for $\hat{v}(z)$ (Aki and Richards, 1980). The second-order WKBJ solution arises when transmission losses are considered.

Equation (5.61) expresses what can be realized in a practical implementation of recursive wavefield extrapolation while equation (5.62) is the theoretical limit of the process. The theoretically correct phase shift from z_1 to z_2 is 2π times the area under the $k_z(z)$ curve between these depths. It turns out that this result is true in one, two or three dimensions but the form of $k_z(z)$ changes. In one dimension, $k_z(z) = f/\hat{v}(z)$ and $\int_0^z k_z(z')dz' = f \int_0^z \hat{v}(z')^{-1} dz' = f\tau(z)$ where $\tau(z)$ is the vertical traveltime. In three dimensions, $k_z(z) = \sqrt{f^2/\hat{v}(z)^2 - k_x^2 - k_y^2}$ and the integral of $k_z(z)$ has no immediate simplification.

An interpretation of the extrapolation phase shift is suggested by incorporating $\sin\theta = \hat{v}k_x/f$ with the result

$$2\pi k_z \Delta z = 2\pi \Delta z \sqrt{\frac{f^2}{\hat{v}^2} - k_x^2} = 2\pi f \frac{\Delta z}{\hat{v}} \sqrt{1 - \left[\frac{\hat{v}k_x}{f}\right]^2} = 2\pi f \tau \cos\theta \tag{5.63}$$

where $\tau = \Delta z/\hat{v}$ has been used. Thus for constant velocity, the extrapolation phase shift is

[2]The name WKBJ is the initials of Wentzel, Kramers, Brilloin, and Jeffreys who all derived it independently. The first three were Quantum theorists while Jeffreys was a geophysicist.

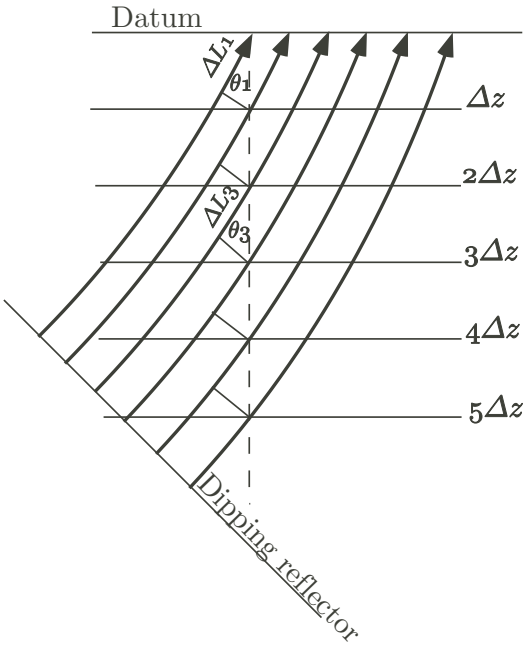

Figure 5.59: The recursive application of the constant-velocity phase shift can model $v(z)$ media as in equation (5.61).

scattering-angle dependent and ranges from a maximum of $2\pi f\tau$ at 0° to a minimum of 0 at 90° . Figure 5.58 shows a geometric interpretation of this phase shift. For a monochromatic plane-wave, the phase shift extrapolator corrects for the phase difference between the wave's arrival at the upper datum at (x_0, z_0) and its arrival at the lower datum at $(x_0, z_0 + \Delta z)$. This phase difference is just 2π times the number of wavelengths that fit in the path difference L, that is

$$\text{phase difference} = 2\pi\frac{L}{\lambda} = 2\pi\frac{\Delta z\cos\theta}{\hat{v}/f} = 2\pi\frac{f\Delta z}{\hat{v}}\cos\theta \tag{5.64}$$

which it the same result as equation (5.63). For a recursive sequence of many constant-velocity phase shifts, the geometric interpretation is as shown in Figure 5.59.

The extrapolation operator is often separated into two terms, one that accomplishes a bulk time shift and another that accomplishes dip-dependent focusing. The bulk time delay operator is simply the extrapolation operator evaluated at $k_x = 0$ (i.e. zero dip) and is called the *static shift operator* or *thin lens operator*. The phase shift it applies is simply $\mu_s = 2\pi f\Delta z/\hat{v}$. Denoting the total phase shift by μ, this suggests that the focusing phase shift is simply $\mu_f = \mu - \mu_s$. Thus,

$$\mu_f = \mu - \mu_s = 2\pi\Delta z\sqrt{\frac{f^2}{\hat{v}^2} - k_x^2} - \frac{2\pi f\Delta z}{\hat{v}} = \frac{2\pi f\Delta z}{\hat{v}}\left[\sqrt{1 - \frac{k_x^2\hat{v}^2}{f^2}} - 1\right] \tag{5.65}$$

where

$$\mu_s = \frac{2\pi f\Delta z}{\hat{v}} \tag{5.66}$$

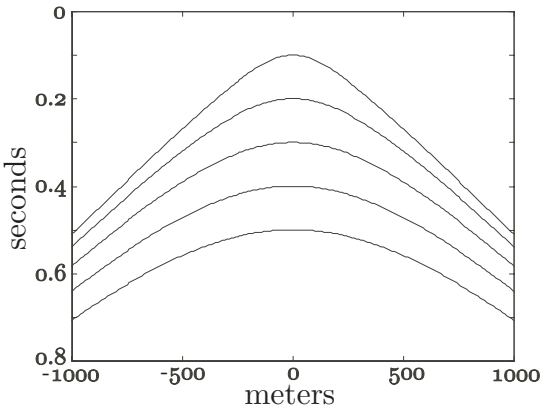

Figure 5.60: A diffraction chart showing the response of five point diffractors as recorded on the surface $z = 0$.

and, in summary,

$$\text{total phase shift} = \mu = \mu_s + \mu_f. \tag{5.67}$$

The focusing phase shift is angle-dependent and vanishes for $k_x = 0$. That the static phase shift accomplishes a bulk time delay can be seen as a consequence of the *phase shift theorem* of digital signal theory (e.g. Karl (1989) page 87). This well-known result from time-series analysis says that a time shift is accomplished in the Fourier domain by a phase shift where the phase is a linear function of frequency. The slope of the linear phase function determines the magnitude of the time shift. Rather than merely quote this result, it is instructive to actually demonstrate it. The static phase shift can be applied to the ERM seismogram by

$$\psi_s(x,t) = \int\limits_{\mathbf{V}_\infty} \phi_0(k_x, f) e^{-\mu_s + 2\pi i(k_x x - ft)} dk_x df. \tag{5.68}$$

Since the static phase shift is independent of k_x, the $k_x$integral can be done directly to give

$$\psi_s(x,t) = \int_{-\infty}^{\infty} \hat{\psi}_0(x,f) e^{-2\pi i(ft + f\Delta z/\hat{v})} df \tag{5.69}$$

where $\hat{\psi}_0(x,f)$ is the temporal Fourier transform of the ERM seismogram. Letting $\tau = \Delta z/\hat{v}$, this becomes

$$\psi_s(x,t) = \int_{-\infty}^{\infty} \hat{\psi}_0(x,f) e^{-2\pi i f(t+\tau)} df = \psi_0(x, t+\tau). \tag{5.70}$$

The last step follows because the integral is an inverse Fourier transform for the time coordinate $t + \tau$. This result is simply a proof of the phase shift theorem in the present context.

Wavefield extrapolation in the space-time domain

Since the extrapolation operator is applied with a multiplication in the Fourier domain, it must be a convolution in the space-time domain. In section 5.2.4 constant-velocity migration was shown to be a nonstationary convolution. Constant-velocity extrapolation, which is a much simpler process

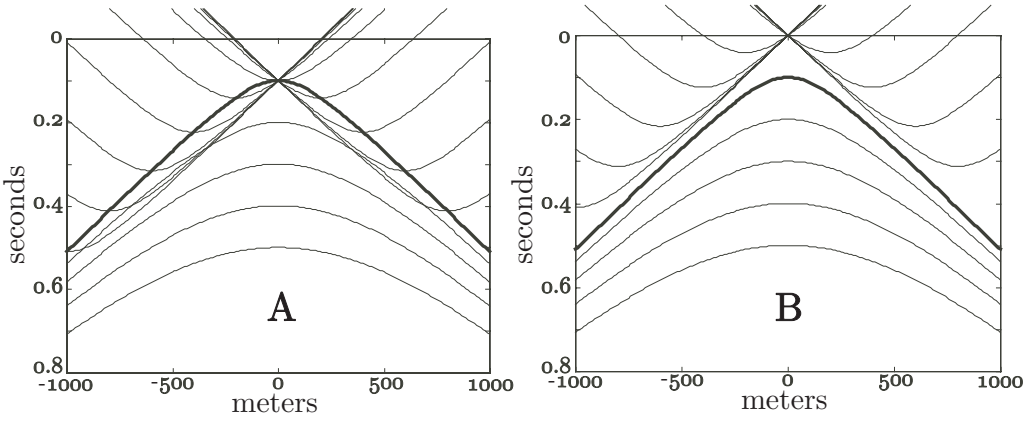

Figure 5.61: The first hyperbola of diffraction chart of Figure 5.60 is shown convolved with the its time reverse. (A) The focusing term only is applied. (B) Both the focusing and the thin-lens term are applied

than migration, is a stationary convolution.

Figure 5.60 shows the response of five different point diffractors, at depths of 200 m, 400 m, 600 m, 800 m, and 1000 m, as recorded at $z = 0$. This diffraction chart was constructed with an exploding reflector velocity $\hat{v} = 2000$ m/s. The chart represents an idealized ERM seismogram for a geology that consists on only five point diffractors. It is desired to determine the space-time shape of the extrapolation operator that will extrapolate this seismogram to 200 m. Since this is the depth of the first diffractor, the first diffraction curve should focus to a point and shift to time zero. The other diffraction curves should all focus somewhat and shift to earlier times by the same amount. In fact, the second hyperbola should be transformed into the first, the third into the second, and so on.

In section 5.2.4 it was seen that replacing each point in the ERM seismogram by a wavefront circle (the *operator*) will focus all hyperbolae at once. With extrapolation, only the first hyperbola should focus and the operator (i.e. the replacement curve) should be the same for all points. Clearly, the operator needs to be concave ($\smile$) to focus the convex ($\frown$) diffraction curves. One process that will focus the first diffraction curve is to *cross correlate* the seismogram with the first diffraction curve itself. To visualize this, imagine tracing the first diffraction curve on a clear piece of film (being careful to mark the coordinate origin on the film) and then placing the apex of this curve at some point on the diffraction chart. The value of the cross correlation is computed by taking the sample-by-sample product of the ERM seismogram and the section represented by the film, and then summing all of the resulting products. Assuming that all amplitudes are either zero (white) or one (black) for simplicity, this reduces to summing the samples in the ERM seismogram that are coincident in space-time with the diffraction curve on the film. This cross-correlation value is then assigned to the location on the ERM seismogram of the coordinate origin on the film. Clearly, when the film is positioned such that its diffraction curve exactly overlies the first diffraction curve, a large value for the cross correlation will result. In fact, since this is the only exact match between the ERM seismogram and the curve on the film, this will be the largest value for the cross correlation. This exact match will occur when the coordinate origin on the film coincides with that on the ERM seismogram.

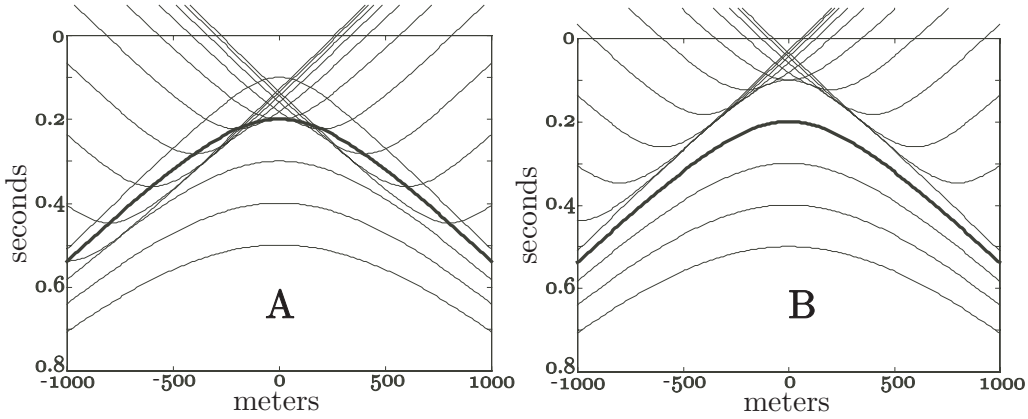

Figure 5.62: The second hyperbola of diffraction chart of Figure 5.60 is shown convolved with the time reverse of the first hyperbola. (A) The focusing term only is applied. (B) Both the focusing and the thin-lens term are applied

Conceptually, the process just described is very simple though it might seem tedious to compute for all samples. Could this cross correlation process be the space-time equivalent of extrapolation? In fact, it is and to realize this it helps to recall from time series analysis that the cross correlation of $a(t)$ with $b(t)$ can be done by time reversing $b(t)$ and convolving. That is $a(t) \otimes b(t) = a(t) \bullet b(-t)$. This result carries over into two dimensions so that the two dimensional cross correlation of the ERM seismogram with the first diffraction curve can be done by reversing the curve in both space and time and then doing a two dimensional convolution. In this case the diffraction curve is symmetric in space (it will not always be though) so the reversal in space does nothing. However, the time reversal flips the first diffraction curve upside down to produce the concave operator that was envisioned.

Figures 5.61 and 5.62 illustrate these concepts with the extrapolation of the ERM seismogram of Figure 5.60 to the depth of the first point diffractor. The extrapolation operator is the time reverse of the first diffraction curve. In Figure 5.61A, only the focusing term is applied to the first diffraction curve with the result that the curve is focused at its apex. In Figure 5.61B, the thin lens term is also included so that the focused diffraction curve is shifted to time zero. In Figure 5.62A, the focusing term is applied to the second diffraction curve and only partial focusing is achieved. When the thin lens term is included in Figure 5.62B, it is apparent that the partially-focused second hyperbola is now equivalent to the first hyperbola.

This perspective of wavefield extrapolation is very powerful and general. The cross correlation argument shows that to extrapolate a wavefield to a particular depth, the response of a point diffractor at that depth as viewed from the current datum must be constructed. Then the wavefield is cross correlated with the diffraction response. The convolution argument is completely equivalent and graphically illustrates the effects of the two parts of the operator: focusing and static shift. It also shows how the extrapolation operator is similar to, but simpler, than the migration operator.

5.4.4 Time and depth migration by phase shift

Thus far, theory has been developed to construct a migrated depth section, $\psi(x, z, t = 0)$ from a zero offset section or ERM seismogram. It is often desired to express the migrated data with a vertical

time coordinate for interpretation. Called τ, this migrated time may be created from z by doing a simple one dimensional stretch using the migration velocity model as discussed in section 5.2.1. It is always correct to do this following migration with $v(x,z)$ with the stretch being defined by

$$\tau(x,z) = \int_0^z \frac{\mathrm{d}z'}{\hat{v}(x,z')} \tag{5.71}$$

A time display, $\psi(x,\tau)$, created from a migrated depth section, $\psi(x,z)$, using equation (5.71) is called a *migrated time* display. Generally this has a meaning distinct from that of *time migration*. The two are equivalent only when lateral velocity variations are absent.

Time migration seeks to create $\psi(x,\tau)$ directly without first creating $\psi(x,z)$. To see how this might be done, recall that the extrapolation phase shift can be written in the (k_x, f) domain as

$$\phi(\Delta z) = \phi_0 e^{i\mu_s + i\mu_f} \tag{5.72}$$

where the static phase shift, μ_s is given by equation (5.66) and the focusing phase shift is given by (5.65). If this extrapolator is applied in a recursive scheme, it is μ_s that progressively moves data to earlier times so that each depth sample can be estimated through the imaging condition $\psi(x,z,t=0)$. If an extrapolation were done with only μ_f then diffractions would focus at their apex time but never move to time zero and flat events (i.e. $k_x = 0$) would never move at all. This is exactly the behavior desired of a time migration. In fact, a time migration can be computed by recursive phase shift using the extrapolation equation

$$\phi(\tau_m + \Delta\tau) = \phi(\tau_m) e^{i\mu_{fm}} \tag{5.73}$$

where μ_{fm} is

$$\mu_f = 2\pi\Delta\tau \left[\sqrt{1 - \frac{k_x^2 \hat{v}_m^2}{f^2}} - 1 \right] \tag{5.74}$$

where $\Delta\tau = \Delta z/\hat{v}_m$ has been used. When computing a time migration by recursive extrapolation, the imaging condition must be changed because data not longer moves to $t = 0$ as it is focused. Instead, focused events are found at $t = \tau$ so that the time migration is given by $\phi(x,\tau,t=\tau)$.

Recursive extrapolation with equation (5.73), or some approximation to it, is a form of time migration while recursive extrapolation with equation (5.72) is a depth migration. There is no essential difference between the two for $v(z)$ because the depth migration can be stretched to time with equation (5.71) or the time migration can be stretched to depth with

$$z(\tau) = \int_0^\tau \hat{v}(x,\tau')\mathrm{d}\tau' \tag{5.75}$$

To see this last point more clearly, if a recursive time migration is formulated as was done for depth migration that leads to equation (5.62), it will be seen that the time migration omits the accumulated phase shift operator

$$e^{i\int_0^z \mu_s(z')\mathrm{d}z'} = \exp\left(i2\pi f \int_0^z \frac{\mathrm{d}z'}{v(z')} \right). \tag{5.76}$$

A stretch of the time migration to depth effectively applies this operator. Time migration is thus equivalent to depth migration for $v(z)$ because the static delay, $\Delta z/\hat{v}(z) = \Delta\tau$, does not depend on x. Since data moves all around during migration following various raypaths, any accumulated time delays in $\psi(x,\tau,t=\tau)$ will be complicated functions of dip and position if $\hat{v}$ varies with x. When $\hat{v}$

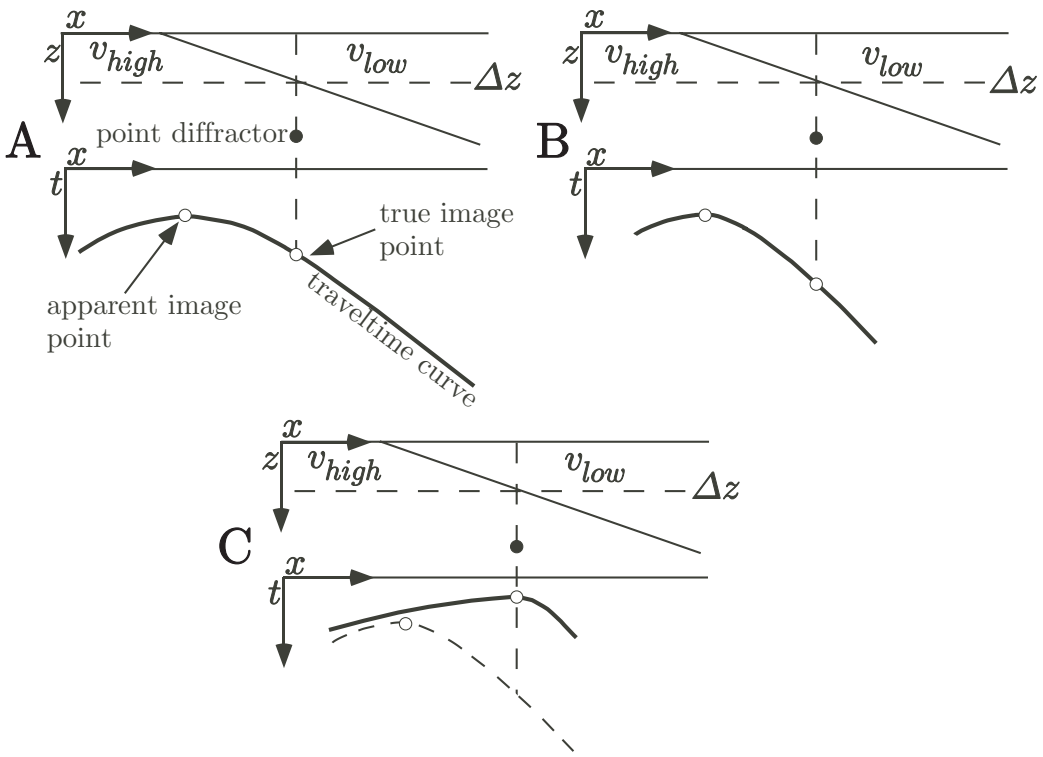

Figure 5.63: (A) A point diffractor sits beneath a low-velocity wedge (top) and its ERM seismogram shows an apparent image point displaced to the left (bottom). (B) After application of the focusing phase shift, the diffraction curve is partially focused but the apparent image point has not moved. (C) The static phase shift moves the apparent image point towards the true image point.

varies only with z, then all data at fixed τ have the same accumulated delay regardless of migration raypath. When $\hat{v}$ varies laterally, time migration actually refracts data in a way that systematically violates Snell's law as was discussed in section 5.2.3.

Figure 5.63A shows the ERM response of a point diffractor in a medium with $\hat{v}(x,z)$. The apparent image point, which is the crest of the traveltime curve, is displaced to the left because velocity is higher there. In Figure 5.63B, the ERM seismogram has been extrapolated to half the depth of the diffractor using the focusing phase shift only. Since μ_f vanishes for $k_x = 0$ it cannot shift the crest of the traveltime curve even if it can apply the lateral velocity variations. All it achieves is a partial focusing of the diffraction response. In Figure 5.63C, the static phase shift has been applied and has had the effect of moving the crest of the traveltime curve to the right towards the true image point. This happens because the time advance of the static shift is greater for lower velocities than for higher ones. Thus it is apparent that extrapolation using μ_s and μ_f can focus the diffractor at the proper location while μ_f alone can never do so.

5.5 Kirchhoff methods

The Fourier methods discussed in section 5.4 are based upon the solution of the scalar wave equation using Fourier transforms. This solution can be derived from a more fundamental approach to partial differential equations called *separation of variables*. An alternative, and equally fundamental, approach to solving the same partial differential equations is based on *Green's theorem* and leads to the family of migration methods known as *Kirchhoff* methods. Though Kirchhoff methods seem superficially quite different from Fourier methods, the uniqueness theorems from partial differential equation theory guarantee that they are equivalent. However, this equivalence only applies to the problem for which both approaches are exact solutions to the wave equation and that is the case of a constant velocity medium, with regular wavefield sampling, and a horizontal recording surface. In all other cases, both methods are implemented with differing approximations and can give very distinct results. Furthermore, even in the exact case, the methods have different computational artifacts.

The wavefront migration methods discussed in section 5.2.4 are simplified examples of Kirchhoff migration. There are two alternative Kirchhoff approaches that differ in that one does the computations on the output space (x, z) and the other does them on the input space (x, t). In the first case, the method of replacing each point of the input section by a wavefront circle, as shown in Figure 5.14, essentially works in the output space. Kirchhoff migration theory provides a detailed prescription for computing the amplitude and phase along the wavefront, and in variable velocity, the shape of the wavefront. The second case essentially sums through the input space along hyperbolic paths to compute each point in the output space. As shown in Figure 5.19, summation along hyperbolae and the superposition of circular wavefronts lead to the same result. Again in this case, Kirchhoff theory shows that the summation along the hyperbola must be done with specific weights and, for variable velocity, how the hyperbola is replaced by a more general shape.

5.5.1 Gauss' theorem and Green's identities

Recall the fundamental theorem of calculus that says

$$\int_a^b \phi'(x)\mathrm{d}x = \phi(x)\,\Big|_a^b = \phi(b) - \phi(a). \tag{5.77}$$

An interpretation of this statement is that the integral of a function ϕ' over the interval $a \le x \le b$ is found by evaluating a different function ϕ at the end points of the interval. The theorem assures us that if ϕ' exists then so does ϕ under a fairly general set of conditions. These functions are of course mathematically related and ϕ is said to be the *integral* of ϕ' or, equivalently, ϕ' is the derivative of ϕ.

Gauss's theorem generalizes this basic result to dimensions greater than one. That is, it says that if the integral over a volume of a certain function is desired, then it can be obtained by evaluating another related function over the surface that bounds the volume. In higher dimensions, the notion of direction is more complicated than the + or - needed in one dimension and vector functions express this. Gauss' theorem is usually written

$$\int_V \vec{\nabla} \cdot \vec{A}\ \mathrm{d}vol = \oint_{\partial V} \vec{A} \cdot \vec{n}\ \mathrm{d}surf. \tag{5.78}$$

Here $\vec{A}$ is a vector function that might physically represent something like fluid flow or electric field and $\vec{n}$ is the *outward pointing* normal to the surface bounding the volume of integration (Figure 5.64).

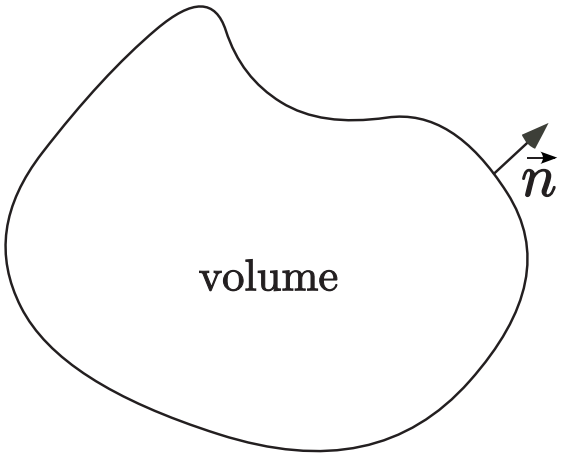

Figure 5.64: Gauss' theorem relates the volume integral of a function $\vec{\nabla} \cdot \vec{A}$ to the surface integral of a related function $\vec{A} \cdot \vec{n}$ where $\vec{n}$ is the outward pointing normal to the surface ∂V bounding the integration volume V.

Equation (5.78) generalizes equation (5.77) in several ways. First, a vector function $\vec{A}$ generalizes the notion of direction where, in one dimension, the sign of ϕ was sufficient. Second, the derivative has been generalized from ϕ' to $\vec{\nabla} \cdot \vec{A}$, and is called the divergence of $\vec{A}$. Finally, the dot product of $\vec{A}$ with $\vec{n}$ integrated over the bounding surface generalizes the simple difference $\phi(b) - \phi(a)$. In the one dimensional case, the outward pointing normal points in the $+x$ direction at b and in the $-x$ direction at a and the surface integral degenerates to a simple sum of two end members.

In many important cases, the vector function $\vec{A}$ can be calculated as the gradient of a scalar potential $\vec{A} = \vec{\nabla}\phi$. In this case Gauss' theorem becomes

$$\int_V \nabla^2\phi \; \mathrm{d}vol = \oint_{\partial V} \frac{\partial \phi}{\partial n} \; \mathrm{d}surf \tag{5.79}$$

where $\nabla^2\phi = \vec{\nabla} \cdot \vec{\nabla}\phi$ and $\frac{\partial \phi}{\partial n} = \vec{\nabla}\phi \cdot \vec{n}$ have been used. The meaning of $\frac{\partial \phi}{\partial n} = \partial_n \phi$ is that it is the component of the vector $\vec{\nabla}\phi$ that is normal to the surface.

Now, return to equation 5.77 and consider the case when $\phi = \phi_1\phi_2$. Then $\phi' = \phi_2\phi_1' + \phi_1\phi_2'$ and

$$\int_a^b [\phi_2\phi_1' + \phi_1\phi_2'] \, \mathrm{d}x = \phi_1\phi_2 \Big|_a^b \tag{5.80}$$

or

$$\int_a^b \phi_2\phi_1' \mathrm{d}x = \phi_1\phi_2 \Big|_a^b - \int_a^b \phi_1\phi_2' \mathrm{d}x \tag{5.81}$$

which is the formula for *integration by parts*. An analogous formula in higher dimensions arises by substituting $A = \phi_2\vec{\nabla}\phi_1$ into equation (5.78). Using the identity $\vec{\nabla} \cdot a\vec{\nabla}b = \vec{\nabla}a \cdot \vec{\nabla}b + a\nabla^2 b$ leads immediately to

$$\int_V \left[\vec{\nabla}\phi_2 \cdot \vec{\nabla}\phi_1 + \phi_2\nabla^2\phi_1\right] \mathrm{d}vol = \oint_{\partial V} \phi_2 \frac{\partial \phi_1}{\partial n} \mathrm{d}surf. \tag{5.82}$$

Then, letting $A = \phi_1 \vec{\nabla} \phi_2$ leads to a similar result

$$\int_V \left[\vec{\nabla} \phi_1 \cdot \vec{\nabla} \phi_2 + \phi_1 \nabla^2 \phi_2 \right] \mathrm{d}vol = \oint_{\partial V} \phi_1 \frac{\partial \phi_2}{\partial n} \mathrm{d}surf. \tag{5.83}$$

Finally, subtracting equation (5.83) from (5.82) results in

$$\int_V \left[\phi_2 \nabla^2 \phi_1 - \phi_1 \nabla^2 \phi_2 \right] \mathrm{d}vol = \oint_{\partial V} \left[\phi_2 \frac{\partial \phi_1}{\partial n} - \phi_1 \frac{\partial \phi_2}{\partial n} \right] \mathrm{d}surf \tag{5.84}$$

which is known as *Green's theorem*.

Green's theorem is fundamental to the derivation of Kirchhoff migration theory. It is a multi-dimensional generalization of the integration by parts formula from elementary calculus and is valuable for its ability to solve certain partial differential equation. At this point, only geometric principles and vector calculus have been involved, the potential physical applications are as yet unspecified. That is, ϕ_1 and ϕ_2 in equation (5.84) are completely arbitrary scalar fields. They may be chosen as desired to conveniently express solutions to a given problem. Typically, in the solution to a partial differential equation like the wave equation, one function is chosen to be the solution to the problem at hand and the other is chosen to be the solution to a simpler *reference problem*. The reference problem is usually selected to have a known analytic solution and that solution is called a *Green's function*.

5.5.2 The Kirchhoff diffraction integral

Let ψ be a solution to the scalar wave equation

$$\nabla^2 \psi(\vec{x}, t) = \frac{1}{v^2} \frac{\partial^2 \psi(\vec{x}, t)}{\partial t^2} \tag{5.85}$$

where the velocity v may depend upon position or it may be constant. To eliminate the time dependence in equation (5.85), let ψ be given by a single Fourier component $\psi(\vec{x}, t) = \hat{\psi}(\vec{x}) e^{-2\pi i f t}$. Then equation (5.85) becomes the *Helmholtz* equation

$$\nabla^2 \hat{\psi}(\vec{x}) = -k^2 \hat{\psi}(\vec{x}) \tag{5.86}$$

where $k^2 = 4\pi^2 f^2 / v^2$. Now, let $g(\vec{x}; \vec{x}_0)$ be the solution to

$$\nabla^2 g(\vec{x}; \vec{x}_0) - k_0^2 g(\vec{x}; \vec{x}_0) = \delta(\vec{x} - \vec{x}_0) \tag{5.87}$$

where $k_0^2 = 4\pi^2 f^2 / v_0^2$ with v_0 constant over all space and $\delta(\vec{x} - \vec{x}_0)$ represents a source at $\vec{x} = \vec{x}_0$. The analytic solution to equation (5.87) is well known (e.g. Morse and Feshbach (1953) page 810) and can be build from a linear combination of the two functions

$$g^{\pm}(\vec{x}; \vec{x}_0) = \frac{e^{\pm i k_0 r}}{r} \tag{5.88}$$

where $r = |\vec{x} - \vec{x}_0|$ and three spatial dimensions are assumed. In two dimensions, the solution must be expressed with Hankel functions that have the asymptotic form

$$g^{\pm}(\vec{x}; \vec{x}_0) \sim \sqrt{\frac{2\pi}{kr}} e^{\pm i k r + i\pi/4}, r \to \infty. \tag{5.89}$$

Since $g(\vec{x};\vec{x}_0)e^{-2\pi ift}$ is the time-dependent Green's function, it is apparent that $g^+ = r^{-1}e^{ik_0r}$ corresponds to a wavefield traveling out from $r = 0$ while $g^- = r^{-1}e^{-ik_0r}$ is a wavefield traveling inward towards $r = 0$. In modelling, g^+ is commonly used and is called the *causal Green's function* while, in migration, it turns out that g^- is appropriate and it is called the *anticausal Green's function*.

Now, apply Greens theorem (equation 5.84) using $\hat{\psi}$ and g^- to get

$$\int_V \left[g^-(\vec{x};\vec{x}_0)\nabla^2\hat{\psi}(\vec{x}) - \hat{\psi}(\vec{x})\nabla^2 g^-(\vec{x};\vec{x}_0) \right] \, \mathrm{d}vol = \oint_{\partial V} \left[g^-(\vec{x};\vec{x}_0)\frac{\partial\hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x})\frac{\partial g^-(\vec{x};\vec{x}_0)}{\partial n} \right] \, \mathrm{d}surf. \tag{5.90}$$

Substituting equations (5.86) and (5.87) into the left hand side of this expression leads to

$$\int_V \left[k^2 - k_0^2\right] g^-(\vec{x};\vec{x}_0)\hat{\psi}(\vec{x}) \, \mathrm{d}vol + \int_V \hat{\psi}(\vec{x})\delta(\vec{x}-\vec{x}_0) \, \mathrm{d}vol = \oint_{\partial V} \left[g^-(\vec{x};\vec{x}_0)\frac{\partial\hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x})\frac{\partial g^-(\vec{x};\vec{x}_0)}{\partial n} \right] \, \mathrm{d}surf \tag{5.91}$$

Assuming that the point $\vec{x}_0$ is interior to the volume V, the delta function collapses the second integral on the left and this expression can be rewritten as

$$\hat{\psi}(\vec{x}_0) = \Lambda(\vec{x}_0) + \oint_{\partial V} \left[g^-(\vec{x};\vec{x}_0)\frac{\partial\hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x})\frac{\partial g^-(\vec{x};\vec{x}_0)}{\partial n} \right] \, \mathrm{d}surf \tag{5.92}$$

where

$$\Lambda(\vec{x}_0) \equiv \int_V \left[k^2 - k_0^2\right] g^-(\vec{x};\vec{x}_0)\hat{\psi}(\vec{x}) \, \mathrm{d}vol. \tag{5.93}$$

Equation (5.92) estimates the wavefield $\hat{\psi}$ at the point $\vec{x}_0$ interior to V as a volume integral plus a surface integral over ∂V. The surface integral is what is desired since we can hope to know $\hat{\psi}$ over the boundary of V. However, the volume integral involves the unknown $\hat{\psi}$ and is essentially not computable. The function $\Lambda(\vec{x}_0)$ expresses this volume integral and can be seen to vanish if the reference medium v_0 is equivalent to the actual medium v over the entire volume. Since g has been chosen as a constant velocity Green's function, Λ can only vanish precisely for constant velocity. However, in the variable velocity case, approximate, ray theoretic Green's functions can be used to help minimize Λ (Docherty, 1991). To the extent that the reference medium does not equal the true medium, then Λ expresses the error in a $\hat{\psi}$ that is computed without Λ. In any case, the next step is to drop Λ and substitute in $g^- = e^{-ik_0r}/r$ into equation (5.92) with the result

$$\hat{\psi}(\vec{x}_0) = \oint_{\partial V} \left[\frac{e^{-ik_0r}}{r}\frac{\partial\hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x})\frac{\partial}{\partial n}\frac{e^{-ik_0r}}{r} \right] \, \mathrm{d}surf. \tag{5.94}$$

The normal derivative of g can now be resolved into two terms

$$\hat{\psi}(\vec{x}_0) = \oint_{\partial V} \left[\frac{e^{-ik_0r}}{r}\frac{\partial\hat{\psi}(\vec{x})}{\partial n} + \frac{ik_0\hat{\psi}(\vec{x})e^{-ik_0r}}{r}\frac{\partial r}{\partial n} + \hat{\psi}(\vec{x})\frac{e^{-ik_0r}}{r^2}\frac{\partial r}{\partial n} \right] \, \mathrm{d}surf. \tag{5.95}$$

Multiplying both sides of this result by $e^{-2\pi ift}$ and recalling that $\psi(\vec{x}_0,t) = \hat{\psi}(\vec{x}_0)e^{-2\pi ift}$ gives

$$\psi(\vec{x}_0,t) = e^{-2\pi ift}\oint_{\partial V} \left[\frac{e^{-ik_0r}}{r}\frac{\partial\hat{\psi}(\vec{x})}{\partial n} + \frac{ik_0\hat{\psi}(\vec{x})e^{-ik_0r}}{r}\frac{\partial r}{\partial n} + \hat{\psi}(\vec{x})\frac{e^{-ik_0r}}{r^2}\frac{\partial r}{\partial n} \right] \, \mathrm{d}surf \tag{5.96}$$

or, using $k_0 = 2\pi f/v_0$,

$$\psi(\vec{x}_0,t) = \oint_{\partial V} \left[\frac{e^{-2\pi i f(t+r/v_0)}}{r} \frac{\partial \hat{\psi}(\vec{x})}{\partial n} + \frac{i2\pi f \hat{\psi} e^{-2\pi i f(t+r/v_0)}}{v_0 r} \frac{\partial r}{\partial n} + \hat{\psi}(\vec{x}) \frac{e^{-2\pi i f(t+r/v_0)}}{r^2} \frac{\partial r}{\partial n} \right] \mathrm{d}surf. \tag{5.97}$$

Now, $\hat{\psi}(\vec{x}) e^{-2\pi i f(t+r/v_0)} = \psi(\vec{x}, t+r/v_0)$ is the wavefield ψ at the point $\vec{x}$ but at the *advanced time* $t + r/v_0$. It is customary to denote this quantity as $[\psi]_{t+r/v_0}$ with the result

$$\psi(\vec{x}_0,t) = \oint_{\partial V} \left[\frac{1}{r} \left[\frac{\partial \psi}{\partial n} \right]_{t+r/v_0} - \frac{1}{v_0 r} \frac{\partial r}{\partial n} \left[\frac{\partial \psi}{\partial t} \right]_{t+r/v_0} + \frac{1}{r^2} \frac{\partial r}{\partial n} [\psi]_{t+r/v_0} \right] \mathrm{d}surf \tag{5.98}$$

where the time derivative in the second term results from $\partial_t \psi = -2\pi i f \psi$. This is a famous result and is known as Kirchhoff's diffraction integral. (In most textbooks this integral is derived for forward modelling with the result that all of the terms are evaluated at the *retarded time* $t - r/v_0$ instead of the advanced time.) It expresses the wavefield at the observation point $\vec{x}_0$ at time t in terms of the wavefield on the boundary ∂V at the advanced time $t + r/v_0$. As with Fourier theory, it appears that knowledge of both ψ and $\partial_n \psi$ are necessary to reconstruct the wavefield at an internal point.

5.5.3 The Kirchhoff migration integral

There are two essential tasks required to convert equation (5.98) into a practical migration formula. First, as mentioned above, the apparent need to know $\partial_n \psi$ must be addressed. Second, the requirement that the integration surface must extend all the way around the volume containing the observation point must be dropped. There are various ways to make both of these arguments that have appeared in the literature. Schneider (1978) dispensed with the need to know $\partial_n \psi$ by using a dipole Green's function with an image source above the recording place. The result was a Green's function that vanished at $z = 0$ and cancelled the $\partial_n \psi$ term in equation (5.98). Schneider also argued that the boundary surface can be a horizontal plane with a hemisphere below, and when the hemisphere is extended to infinity, contributions from it vanish. Wiggins (1984) adapted Schneider's technique to rough topography. Docherty (1991) showed that a monopole Green's function, as used here in equation (5.88) can lead to the accepted result and also challenged Schneider's argument that the integral over the infinite hemisphere can be neglected. Instead, Docherty formulated the expression with the *backscattered* wavefield received at the surface and simply put the lower part of the integration surface beneath the reflector. On physical grounds, the wavefield beneath the reflector is not expected to contain significant information about the backscattered field and so it may be neglected. In reality, it does contain some information because the reflected and transmitted wavefields are related through boundary conditions on the reflector, but these are subtle second-order effects. It should also be recalled that virtually all authors on this subject (the present writer included) have approached it with knowledge of the desired result. After all, migration by summation along diffraction curves or by wavefront superposition has been done for many years. Though Schneider's derivation has been criticized, his final expressions are considered correct.

As a first step in adapting equation (5.98) it is usually considered appropriate to discard the term $\frac{1}{r^2} \frac{\partial r}{\partial n} [\psi(\vec{x})]_{t+r/v_0}$. This is called the *near-field term* and decays more strongly with r than the other two terms. Then, the surface $S = \partial V$ will be taken as the $z = 0$ plane, S_0, plus the surface infinitesimally below the reflector, S_z, and finally these surfaces will be joined at infinity by vertical cylindrical walls, S_∞, (Figure 5.65). As mentioned previously, the integration over S_z is not expected to contribute significantly to reconstruction of the backscattered field. Also, the integration over

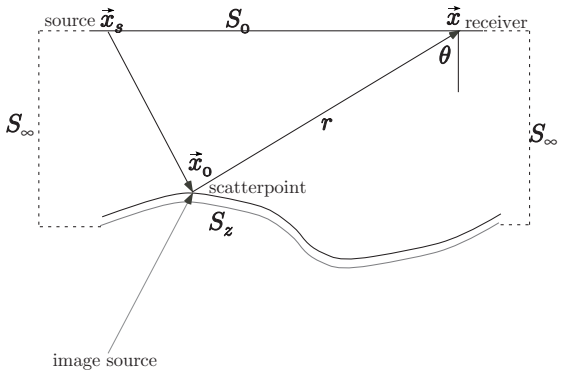

Figure 5.65: The geometry for Kirchhoff migration is shown. The integration surface is $S_0 + S_z + S_\infty$ and it is argued that only S_0 contributes meaningfully to the estimation of the backscattered field at vx_0.

S_∞, though it may contribute, can never be realized due to finite aperture limitations and its neglect may introduce unavoidable artifacts. With these considerations, equation (5.98) becomes

$$\psi(\vec{x}_0, t) = \oint_{S_0} \left[\frac{-1}{r} \left[\frac{\partial \psi}{\partial z} \right]_{t+r/v_0} + \frac{1}{v_0 r} \frac{\partial r}{\partial z} \left[\frac{\partial \psi}{\partial t} \right]_{t+r/v_0} \right] \mathrm{d}surf \tag{5.99}$$

where the signs on the terms arise because $\vec{n}$ is the outward normal and z is increasing downward so that $\partial_n = -\partial_z$.

Now, $\partial_z \psi$ must be evaluated. Figure 5.65 shows the source wavefield being scattered from the reflector at $\vec{x}_0$ which is called the *scatterpoint*. A simple model for ψ is that it is approximately the wavefield from a point source, placed at the image source location, that passes through the scatterpoint to the receiver. This can be expressed as

$$\psi(\vec{x}, t) \sim \frac{1}{r} A(t - r/v) = \frac{[A]_{t-r/v}}{r} \tag{5.100}$$

where $A(t)$ is the source waveform at the scatterpoint. Using the chain rule gives

$$\frac{\partial \psi}{\partial z} = \frac{\partial r}{\partial z} \frac{\partial \psi}{\partial r} = \frac{\partial r}{\partial z} \left[\frac{-1}{vr} \left[\frac{\partial A}{\partial t} \right]_{t-r/v} - \frac{[A]_{t-r/v}}{r^2} \right]. \tag{5.101}$$

If the second term is neglected (a near-field term), this becomes

$$\frac{\partial \psi}{\partial z} = \frac{\partial r}{\partial z} \frac{-1}{vr} \left[\frac{\partial A}{\partial t} \right]_{t-r/v} = -\frac{\partial r}{\partial z} \frac{1}{v} \frac{\partial \psi}{\partial t}. \tag{5.102}$$

When this is substituted into equation 5.99, the two terms in square brackets become similar both involving the time derivative of the advanced wavefield. These will combine if v_0 is now taken to be the same as v. Thus

$$\psi(\vec{x}_0, t) = \oint_{S_0} \frac{2}{vr} \frac{\partial r}{\partial z} \left[\frac{\partial \psi}{\partial t} \right]_{t+r/v} \mathrm{d}surf \tag{5.103}$$

Finally, consider $\partial_z r$. Since $r = \sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}$ this can be written

$$\frac{\partial r}{\partial z} = \frac{\partial}{\partial z}\sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2} = \frac{z}{r} = \cos\theta \tag{5.104}$$

where θ is the vertical angle between the receiver location and ray to the scatterpoint. With this, the final formula for the scattered wavefield just above the reflector is

$$\psi(\vec{x}_0, t) = \oint_{S_0} \frac{2\cos\theta}{vr}\left[\frac{\partial\psi}{\partial t}\right]_{t+r/v} \mathrm{d}surf \tag{5.105}$$

Equation (5.105) is not yet a migration equation. As mentioned, it provides an estimate of the scattered wavefield just above the scatterpoint. Thus it is a form of wavefield extrapolation though it is direct, not recursive. A migration equation must purport to estimate reflectivity, not just the scattered wavefield and for this purpose a model relating the wavefield to the reflectivity is required. The simplest such model is the exploding reflector model (section 5.2.6) which asserts that the reflectivity is identical to the downward continued scattered wavefield at $t = 0$ provided that the downward continuation is done with $\hat{v} = v/2$. Thus, an ERM migration equation follows immediately from equation (5.105) as

$$\psi(\vec{x}_0, 0) = \oint_{S_0} \frac{2\cos\theta}{\hat{v}r}\left[\frac{\partial\psi}{\partial t}\right]_{r/\hat{v}} \mathrm{d}surf = \oint_{S_0} \frac{4\cos\theta}{vr}\left[\frac{\partial\psi}{\partial t}\right]_{2r/v} \mathrm{d}surf. \tag{5.106}$$

This result, derived by many authors including Schneider (1978) and Scales (1995), expresses migration by summation along hyperbolic travelpaths through the input data space. The hyperbolic summation is somewhat hidden by the notation but is indicated by $[\partial_t\psi]_{2r/v}$. Recall that this notation means that the expression in square brackets is to be evaluated at the time indicated by the subscript. That is, as $\partial_t\psi(\vec{x}, t)$ is integrated over the $z = 0$ plane, only those specific traveltimes values are selected that obey

$$t = \frac{2r}{v} = \frac{2\sqrt{(x-x_0)^2 + (y-y_0)^2 + z_0^2}}{v} \tag{5.107}$$

which is the equation of a zero-offset diffraction hyperbola. When squared, this result is a three dimensional version of equation (5.36).

In addition to diffraction summation, equation (5.106) requires that the data be scaled by $4\cos\theta/(vr)$ and that the time derivative be taken before summation. These additional details were not indicated by the simple geometric theory of section 5.2.4 and are major benefits of Kirchhoff theory. It is these sort of corrections that are necessary to move towards the goal of creating bandlimited reflectivity. The same correction procedures are contained implicitly in f-k migration.

The considerations taken in deriving equation (5.106) suggest why ERM migration does not achieve correct amplitudes. As mentioned following equation (5.105), a model linking the backscattered wavefield to reflectivity was required. A more physical model will illustrate the shortcomings of the exploding reflector model. Such a model has been advanced by Berkhout (1985) and others. They consider that the source wavefield propagates directly to the reflector, undergoing transmission losses and geometrical spreading but without multiples and converted modes. At the reflector it is scattered upward with an angle-dependent reflection coefficient and then propagates upward to the receivers, with further geometrical spreading and transmission losses but again without multiples and mode conversions. This allows an interpretation of equation (5.105) as equivalent to the wave-

field propagated from the source to the scatterpoint and scaled by the reflection coefficient. Thus, a better way to estimate the reflectivity is to produce a model of the source wavefield as propagated down to the scatterpoint and divide the result of equation (5.105) by this modeled wavefield. This is a very general imaging condition called the *deconvolution imaging condition* that works for pre-stack and zero offset data. The ERM imaging condition is kinematically correct but does not achieve the same amplitudes as the deconvolution imaging condition.

Kirchhoff migration is one of the most adaptable migration schemes available. It can be easily modified to account for such difficulties as topography, irregular recording geometry, pre-stack migration, and converted wave imaging. When formulated as a depth migration, it tends to be a slow method because great care must be taken in the raytracing (Gray, 1986). When formulated as a time migration, straight-ray calculations using RMS velocities can be done to greatly speed the process. Another advantage of Kirchhoff methods is the ability to perform "target-oriented" migrations. That is, equation (5.106) need only be evaluated for those points in (x,z) which comprise the target. Since the cost of computation is directly proportional to the number of output points, this can greatly reduce the run times and makes migration parameter testing very feasible.

5.6 Finite difference methods

5.6.1 Finite difference extrapolation by Taylor series

Finite difference techniques are perhaps the most direct, but often the least intuitive, of migration methods. As the name implies, they involve the approximation of analytic derivatives of the wave equation with finite difference expressions. To see how this might be done, consider the definition of the derivative of an arbitrary function $\psi(z)$

$$\frac{\mathrm{d}\psi(z)}{\mathrm{d}z} = \lim_{\Delta z \to 0} \frac{\psi(z+\Delta z) - \psi(z)}{\Delta z}. \tag{5.108}$$

A simple finite difference approximation for this derivative simply involves omitting the limit

$$\frac{\mathrm{d}\psi(z)}{\mathrm{d}z} \approx \delta_z^{1+}\psi(z) \equiv \frac{\psi(z+\Delta z) - \psi(z)}{\Delta z}. \tag{5.109}$$

δ_z^{1+} is the *first-order forward-difference operator*. Here Δz is assumed small with respect to length scales of interest (i.e. wavelengths) but is still finite.

Finite difference operators can be used to predict or extrapolate a function. Suppose that values for $\psi(z)$ and its first derivative are known at z, then equation 5.109 can be rearranged to predict $\psi(z+\Delta z)$

$$\psi(z+\Delta z) \approx \psi(z) + \frac{\mathrm{d}\psi(z)}{\mathrm{d}z}\Delta z. \tag{5.110}$$

This can be regarded as a truncated Taylor series. Taylor's theorem provides a method for extrapolation of a function provided that the function and all of its derivatives are known at a single point

$$\psi(z+\Delta z) = \psi(z) + \frac{\mathrm{d}^1\psi(z)}{\mathrm{d}z^1}\Delta z + \frac{1}{2}\frac{\mathrm{d}^2\psi(z)}{\mathrm{d}z^2}\Delta z^2 + \frac{1}{6}\frac{\mathrm{d}^3\psi(z)}{\mathrm{d}z^3}\Delta z^3 + \ldots. \tag{5.111}$$

If all derivatives exist, up to infinite order at z, then there is no limit to the extrapolation distance. Essentially, if the function and all its derivatives are known at one location, then it is determined everywhere.

If finite difference extrapolation is equivalent to using a truncated Taylor series, then what relation does wavefield extrapolation by phase shift have with Taylor series? Recall that if $\phi(z)$ is a solution to the scalar wave equation in the Fourier domain, then it can be extrapolated by

$$\phi(z+\Delta z) = \phi(z) e^{2\pi i k_z \Delta z}. \tag{5.112}$$

Here, $k_z$is the vertical wavenumber whose value may be determined from the scalar wave dispersion relation, $k_z = \sqrt{f^2/v^2 - k_x^2}$. Furthermore, if $\phi_0(k_x, f)$ is the (k_x, f) spectrum of data measured at $z = 0$, then, for constant velocity, $\phi(z)$ is given by

$$\phi(z) = \phi_0 e^{2\pi i k_z z}. \tag{5.113}$$

As an illustration of the connection between Taylor series and the phase-shift extrapolator, consider using Taylor series applied to equation (5.113) to derive something equivalent to equation (5.112). From equation (5.113) the various derivatives can be estimated with the result that the n$^{\text{th}}$ derivative is

$$\frac{\mathrm{d}^n \phi(z)}{\mathrm{d}z^n} = [2\pi i k_z]^n \phi(z). \tag{5.114}$$

Then, using this result, a Taylor series expression for the extrapolation of $\phi(z)$ to $\phi(z+\Delta z)$ is

$$\phi(z+\Delta z) = \phi(z) + [2\pi i k_z]\phi(z)\Delta z + \frac{1}{2}[2\pi i k_z]^2 \phi(z)\Delta z^2 + \frac{1}{6}[2\pi i k_z]^3 \phi(z)\Delta z^3 + \dots \tag{5.115}$$

or

$$\phi(z+\Delta z) = \phi(z)\left[1 + 2\pi i k_z \Delta z + \frac{1}{2}[2\pi i k_z \Delta z]^2 + \frac{1}{6}[2\pi i k_z \Delta z]^3 + \dots\right]. \tag{5.116}$$

At this point, recall the expression for the series expansion of an exponential is $e^x = 1 + x + x^2/2 + x^3/6 + \dots$ which affords the conclusion that the infinite series in square brackets of equation (5.116) may be summed to obtain $\left[1 + 2\pi i k_z \Delta z + \frac{1}{2}[2\pi i k_z \Delta z]^2 + \frac{1}{6}[2\pi i k_z \Delta z]^3 + \dots\right] = e^{2\pi i k_z \Delta z}$. Thus, if infinitely many terms are retained in the series, equation (5.116) is equivalent to equation (5.112). It may be concluded that wavefield extrapolation by phase shift is equivalent to extrapolation with an infinite-order Taylor series and there is no upper limit on the allowed size of Δz (in constant velocity). Alternatively, wavefield extrapolation by finite difference approximations is equivalent to extrapolation with a truncated Taylor series and the Δz step size will have a definite upper limit of validity.

5.6.2 Other finite difference operators

The finite difference approximation used in equation (5.109) is the simplest possible and there are many others. The subject of solving partial differential equations by finite difference methods has produced a vast literature while only a brief examination is possible here. Aki and Richards (1980), Ames (1992), and Durran (1999) contain much more thorough discussions.

The forward difference operator in (5.109) uses the points at z and $z+\Delta z$. A *backward difference operator* is equally acceptable as an approximate derivative

$$\frac{\mathrm{d}\psi(z)}{\mathrm{d}z} \approx \delta_z^{1-}\psi(z) \equiv \frac{\psi(z) - \psi(z-\Delta z)}{\Delta z}. \tag{5.117}$$

Both of these expressions give similar problems in practice that are related to the fact that the difference is not centered at the estimation point. That is $\psi(z+\Delta z) - \psi(z)$ is centered at $z + \Delta z/2$

while the backward difference is centered at $z - \Delta z/2$. This suggests forming a *centered difference* by averaging the forward and backward operators

$$\frac{\mathrm{d}\psi(z)}{\mathrm{d}z} \approx \frac{1}{2}\left[\delta_z^{1+} + \delta_z^{1-}\right]\psi(z) \equiv \frac{\psi(z+dz) - \psi(z-\Delta z)}{2\Delta z} \equiv \delta_z^1. \tag{5.118}$$

The centered difference is usually superior to the forward and backward differences and should be used whenever possible.

An approximation for the second derivative can be developed by applying the forward and backward operators in succession

$$\frac{\mathrm{d}^2\psi(z)}{\mathrm{d}z^2} \approx \delta_z^{1+}\delta_z^{1-}\psi(z) = \delta_z^{1+}\left[\frac{\psi(z) - \psi(z-\Delta z)}{\Delta z}\right] = \frac{\psi(z+\Delta z) - \psi(z)}{\Delta z^2} - \frac{\psi(z) - \psi(z-\Delta z)}{\Delta z^2} \tag{5.119}$$

or

$$\frac{\mathrm{d}^2\psi(z)}{\mathrm{d}z^2} \approx \delta_z^2\psi(z) \equiv \frac{\psi(z+\Delta z) - 2\psi(z) - \psi(z-\Delta z)}{\Delta z^2} \tag{5.120}$$

This is a centered approximation to the second derivative.

5.6.3 Finite difference migration

Most finite difference migrations methods are formulated in the space-time domain though space-frequency methods are also common. Also, they are all recursive extrapolation techniques. Though there are many equivalent ways to develop any particular method, a consistent approach can be formulated by beginning in the frequency domain with phase shift theory as follows:

1) Develop a suitable rational approximation to the one way dispersion relation for scalar waves. A rational approximation is one which eliminates the square root though it may involve multiplication, division, and powers of frequencies and wavenumbers.

2) Construct a space-time domain differential equation from the approximate dispersion relation by the replacement rules

$$f \to \frac{i}{2\pi}\frac{\partial}{\partial t}; k_x \to \frac{-i}{2\pi}\frac{\partial}{\partial x}; k_y \to \frac{-i}{2\pi}\frac{\partial}{\partial y}; k_z \to \frac{-i}{2\pi}\frac{\partial}{\partial z}; \tag{5.121}$$

3) Choose an appropriate form for the finite difference operators (i.e. forward, backward,or central differences).

4) Develop the difference equation which corresponds to the differential equation found in step 2.

5) Solve the difference equation for the extrapolated wavefield.

6) Implement a migration by recursive extrapolation computer algorithm using step 5.

As an illustration, the *15 degree* finite difference time migration algorithm will be developed. This was one of the first migration algorithms to be developed and is described in Claerbout (1976). Recall that in the Fourier domain, time migration by recursive extrapolation proceeds by using the

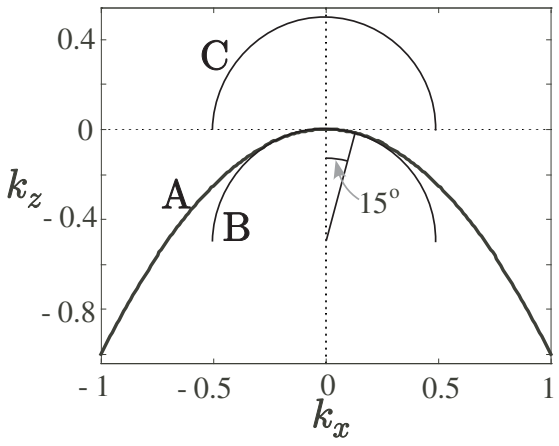

Figure 5.66: For a particular f and $\hat{v}$, (A) the dispersion relation (equation (5.125)) of the parabolic wave equation; (B) the exact dispersion relation (equation (5.123)) of the focusing phase shift; (C) the full dispersion relation of the scalar wave equation.

focusing phase shift at each extrapolation step

$$\mu_f = \frac{2\pi f \Delta z}{\hat{v}} \left[\sqrt{1 - \frac{k_x^2 \hat{v}^2}{f^2}} - 1 \right]. \tag{5.122}$$

Since the general form of an extrapolation phase shift is $phase = 2\pi k_z \Delta z$, a finite difference time-migration requires a "wave equation" whose dispersion relation approximates

$$\tilde{k}_z = \frac{f}{\hat{v}} \left[\sqrt{1 - \frac{k_x^2 \hat{v}^2}{f^2}} - 1 \right]. \tag{5.123}$$

where $\tilde{k}_z$ is used rather than k_z to reserve the latter term for the real vertical wavenumber. It is not acceptable to perform the substitutions of relation (5.121) into this result because the square root of a differential operator results and that is difficult to work with. Therefore, a rational approximation to the square root in equation (5.123) is desired. The second term in the square root is $\sin^2\theta$ where θ is the scattering angle. For small angles (i.e. energy traveling nearly vertically) it is expected that an approximate form can be obtained by expanding the square root and keeping only the first few terms

$$\sqrt{1 - \frac{k_x^2 \hat{v}^2}{f^2}} = 1 - \frac{k_x^2 \hat{v}^2}{2f^2} + \frac{3k_x^2 \hat{v}^2}{8f^2} + \ldots. \tag{5.124}$$

Truncating equation (5.124) at two terms and substituting the result into equation (5.123) gives

$$\tilde{k}_z \approx \frac{f}{\hat{v}} \left[1 - \frac{k_x^2 \hat{v}^2}{2f^2} - 1 \right] = -\frac{k_x^2 \hat{v}^2}{2f}. \tag{5.125}$$

To construct a partial differential equation from this approximate dispersion, first multiply both sides by $2f/\hat{v}$ and replace the spectral multiplications with partial derivatives according to relation

(5.121). This results in

$$\frac{2}{\hat{v}}\frac{\partial^2\psi(x,\tilde{z},t)}{\partial\tilde{z}\partial t}+\frac{\partial^2\psi(x,\tilde{z},t)}{\partial x^2}=0. \tag{5.126}$$

This is known as the *15°* or *parabolic* wave equation. The reason for this name is suggested by Figure 5.66 which shows that equation (5.125) is a good approximation to equation (5.123) for angles less than about 15° .

Before proceeding with the finite difference approximations, it is helpful to change variables in equation (5.126) from $\tilde{z}$ to τ where $\tau=\tilde{z}/\hat{v}$. The $\tilde{z}$ derivative changes according to $\partial_{\tilde{z}}=(\partial_{\tilde{z}}\tau)\partial_\tau=\hat{v}^{-1}\partial_\tau$. This allows equation (5.126) to be recast as

$$\frac{2}{\hat{v}^2}\frac{\partial^2\psi(x,\tau,t)}{\partial\tau\partial t}+\frac{\partial^2\psi(x,\tau,t)}{\partial x^2}=0. \tag{5.127}$$

Equation (5.127) is an approximate wave equation which can be used to migrate stacked data from zero offset time to migrated time. Though this equation is rarely used anymore, its finite difference implementation illustrates most of the features of the method and is simpler than higher order methods.

The finite difference approximation to equation (5.127) is well described by Claerbout (1976) and that approach is followed here. Let

$$\frac{1}{\Delta x^2}T=\delta_x^2\approx\frac{\partial^2}{\partial x^2}\quad\text{and}\quad\psi_j^k=\psi(x,k\Delta\tau,j\Delta t). \tag{5.128}$$

Then the x derivatives of equation (5.127) are approximated as

$$\delta_\tau\delta_t\psi_j^k=-\frac{\hat{v}^2}{2\Delta x^2}T\psi_j^k \tag{5.129}$$

where δ_τ and δ_t are, as yet, unspecified finite difference operators. The δ_t operator is implemented as a forward difference but the right-hand-side of equation (5.129) is also modified to ensure that both sides of the equation are centered at the same grid location. The result is

$$\delta_\tau\left[\psi_{j+1}^k-\psi_j^k\right]=-\frac{\Delta t\hat{v}^2}{4\Delta x^2}T\left[\psi_{j+1}^k+\psi_j^k\right] \tag{5.130}$$

where the right-hand-side represents the average of T applied to two grid points. This process of maintaining grid balance is essential in producing a stable algorithm and is a *Crank-Nicholson* method. Next the τ operator is implemented in a similar way as a balanced forward difference with the result

$$\left[\psi_{j+1}^{k+1}-\psi_{j+1}^k\right]-\left[\psi_j^{k+1}-\psi_j^k\right]=-\frac{\Delta\tau\Delta t\hat{v}^2}{8\Delta x^2}T\left[\psi_{j+1}^{k+1}+\psi_{j+1}^k+\psi_j^{k+1}+\psi_j^k\right]. \tag{5.131}$$

Finally, to downward continue, equation (5.131) must be solved for ψ_j^{k+1}. This can be written as

$$\left[I-aT\right]\psi_j^{k+1}=\left[I+aT\right]\left[\psi_{j+1}^{k+1}+\psi_j^k\right]-\left[I-aT\right]\psi_{j+1}^k \tag{5.132}$$

where $a=\Delta\tau\Delta t\hat{v}^2/8\Delta x^2$. Equation (5.132) is solved numerically for the unknown ψ_j^{k+1} where it is assumed that all of the quantities on the right-hand-side are known. This is an example of an *implicit* finite difference method because it requires the numerical inversion of the matrix operator $I+aT$.

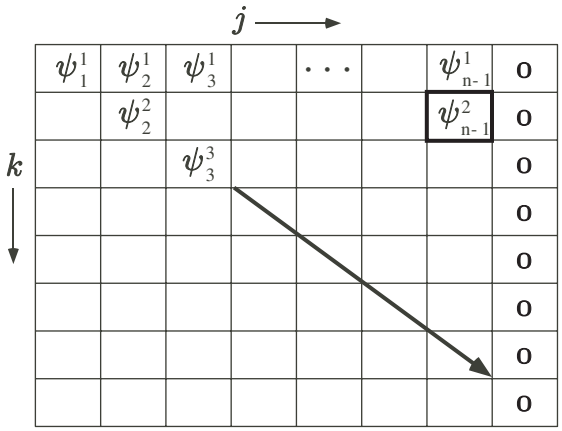

Figure 5.67: The parabolic wave equation is solved on a 2D grid using finite differences. The first row contains the known data and the last column is set to zero. The solution proceeds row-by-row and begins in the upper right corner.

The differencing procedure can be viewed on a (j, k) grid as shown in Figure 5.67. The x axis is orthogonal to this figure and is not shown. The first row contains the measured CMP stack and the last column (corresponding to maximum time) is set to zero. Consider the four grid points in the upper right corner. Three of these are prescribed by initial conditions and the fourth, ψ^1_{n-1} can then be calculated by equation (5.132). Subsequently, ψ^1_{n-2} can be solved for and so on until the entire second row in calculated. Computation then moves to the third row and continues until the entire grid is completed. Each row corresponds to an extrapolated τ section and the final migrated section corresponds to $t = \tau$ which is the diagonal.

From this example, many of the well known characteristics of finite difference migration algorithms are easily deduced such as

- They are recursive extrapolation schemes.
- They use an approximate form of the exact dispersion relation for scalar waves. This means that they are all dip or scattering angle limited.
- They are very flexible in handling velocity variations since the velocity may simply be varied with each grid cell.
- The use of finite difference derivatives means that even the approximate dispersion relation is not exactly realized.
- Finite difference derivatives are not unique and the accuracy of a method depends strongly on the sophistication of the approximations used. Generally, finite difference methods need many more samples per wavelength that Fourier methods (6-10 versus 2-3).
- They can be either time or depth migrations.
- They can be posed in the space-time or space-frequency domain.

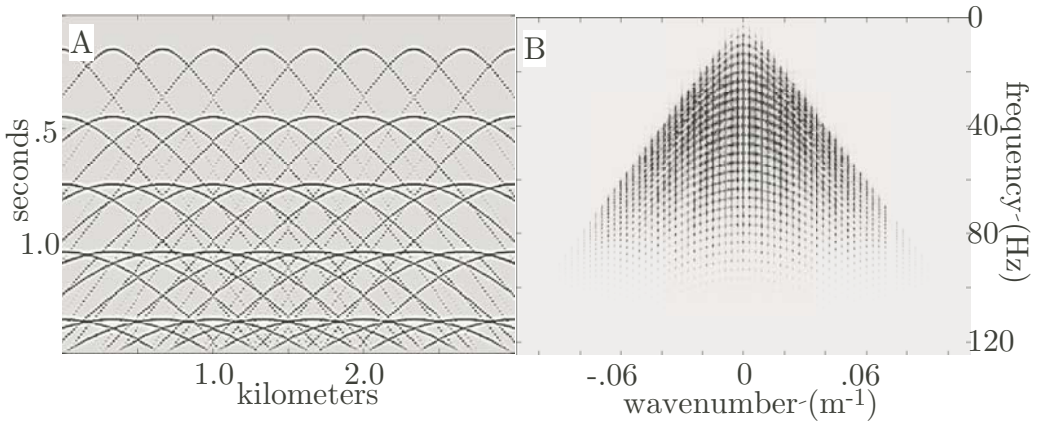

Figure 5.68: (A) An ERM seismogram that models a regular grid of point diffractors. (B) The (k_x, f) spectrum of (A).

5.7 Practical considerations of finite datasets

From the perspective of f-k migration theory, optimizing the ability of seismic data to resolve earth features requires maximization of the spectral bandwidth after migration. The k_x (horizontal wavenumber) bandwidth determines the lateral resolution and is directly proportional to the maximum frequency and the sine of the maximum scattering angle and inversely proportional to velocity. Constraints on the maximum scattering angle can be derived by examining the three effects of finite spatial aperture, finite recording time, and discrete spatial sampling. These effects are analyzed, assuming zero-offset recording, for the case of a linear (constant gradient) $v(z)$ medium. Explicit analytic expressions are derived for the limits imposed on scattering angle for each of the three effects. Plotting these scattering angle limits versus depth limits for assumed recording parameters is an effective way to appreciate their impact on recording. When considered in context with f-k migration theory, these scattering angle limits can be seen to limit spatial resolution and the possibility of recording specific reflector dips. Seismic surveys designed with the linear $v(z)$ theory are often much less expensive than constant velocity theory designs.

Seismic line length and maximum record time place definite limits on the maximum scattering angle that can be recorded, and hence imaged, on a migrated zero-offset section. Since horizontal resolution depends directly on the sine of the maximum scattering angle (Vermeer (1990) and many others), it is important to understand these effects for survey design and interpretation. Furthermore, the observation of a normal incidence reflection from a dipping reflector requires having a scattering angle spectrum whose limits exceed the reflector dip.

The imposition of finite recording apertures in space and time actually imprints a strong spatial-temporal variation (i.e. nonstationarity) on the spectral content of a migrated section. As an example, consider the synthetic seismic section shown in Figure 5.68A. This shows a zero offset (constant velocity) simulation of the response of a grid of point scatterers (diffractors) distributed uniformly throughout the section. Note how the diffraction responses change geometry and how the recording apertures truncate each one differently. Figure 5.68B is a display of the (k_x, f) amplitude spectrum for this section. As expected from elementary theory, all energy is confined to a triangular

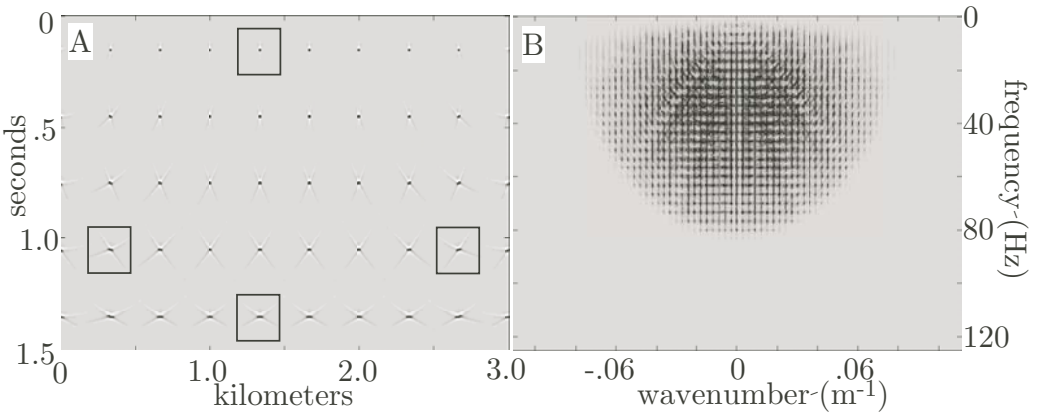

Figure 5.69: (A) The f-k migration of the ERM seismogram of Figure 5.68A. (B) The (k_x, f) spectrum of (A).

region defined by $|k_x| < f/\hat{v}$.

Figure 5.69A shows this section after a constant velocity f-k migration and Figure 5.69B is the (k_x, f) spectrum after migration. The spectrum shows the expected behavior in that the triangular region of Figure 5.68B has unfolded into a circle. Essentially, each frequency (horizontal line in Figure 5.68B) maps to a circle in Figure 5.69B (see Chun and Jacewitz (1981) for a discussion). Note that f-k migration theory as usually stated Stolt (1978) assumes infinite apertures while close inspection of the focal points in Figure 5.69A shows that their geometry varies strongly with position.

The four focal points shown boxed in Figure 5.69A are enlarged in Figure 5.70A. Considering the focal points near the center of the spatial aperture, a small, tight focal point at the top of the section grades to a broad, dispersed smear near the bottom. Alternatively, assessing at constant time shows the focal points grading from strongly asymmetric-left through symmetric to asymmetric-right. Figure 5.70B shows local (k_x, f) spectra of the four focal points in 5.70A. Comparing with Figure 5.69B shows that these local spectra are dramatically different from the global spectrum. Only the top-center point has the full circular spectrum expected from the infinite aperture theory while the others show strong asymmetry or severe bandwidth restrictions. These local spectra determine the local resolution characteristics of the aperture limited seismic section. Schuster (1997) gives a formal theory (assuming constant velocity) for these focal points and shows that the local spectra are bounded by scattered rays that extend from the scatter point to either side of the section. This current paper shows how to estimate these scattering angles in a realistic setting and therefore how to assess resolution implications.

For constant velocity, the computation of limiting scattering angles is well understood but this approach often results in overly expensive survey designs. An analysis with a constant velocity gradient is much more realistic as it allows for first order effects of ray bending by refraction. Such an analysis is presented here together with a simple graphical method of assessing the results.

5.7.1 Finite dataset theory

Stolt (1978) established f-k migration theory for the post-stack, zero-offset case. A fundamental result is that constant velocity migration is accomplished by a mapping from the (k_x, f) plane to

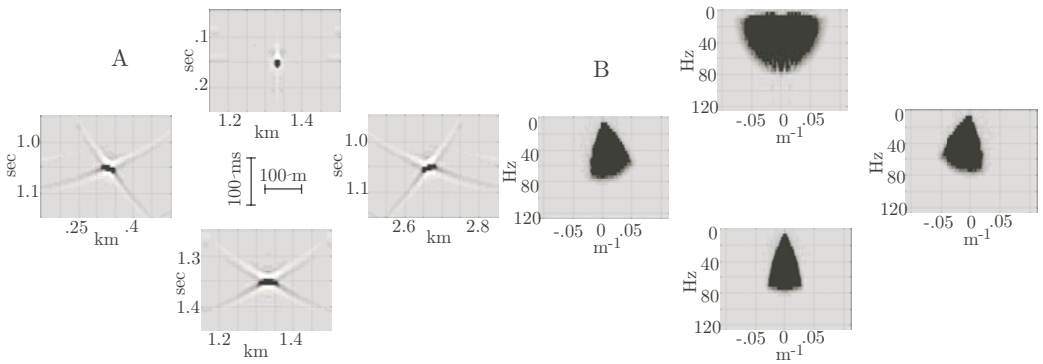

Figure 5.70: (A) Enlargements of the boxed focal points of Figure 5.69A. (B) Local (k_x, f) spectra of the focal points in (A).

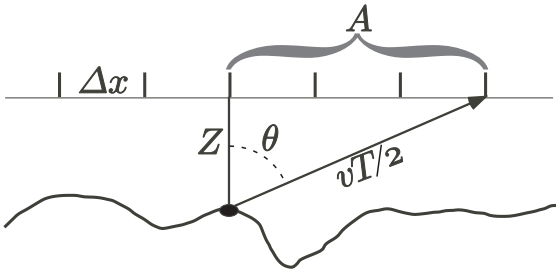

Figure 5.71: A ray from a scatterpoint on a reflector requires a certain aperture A and record length T to be captured. Also, the CMP sample rate Δx must be sufficiently small to avoid spatial aliasing.

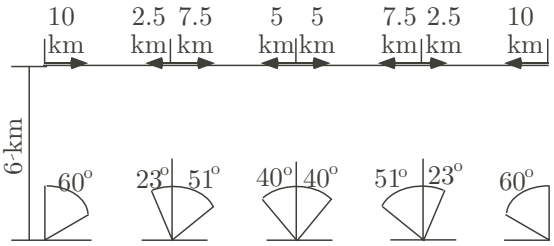

Figure 5.72: A fixed length seismic line induces an aperture effect that varies laterally along the line. In turn, this limits the scattering angle spectrum.

the (k_x, k_z) plane as was discussed in section 5.4.1 and illustrated in Figure 5.49. As can be deduced from the figure, the k_x bandwidth after migration is limited by

$$k_{xlim} = k_{xmax} = \frac{2f_{max}\sin(\theta_{max})}{v}. \tag{5.133}$$

$$k_z^2 = \frac{f^2}{(v/2)^2} - k_x^2 \tag{5.134}$$

$$\lambda_{min} = \lambda_{dom} = 2.85\lambda_{min} = \frac{\lambda_{dom}}{2} = 1.425\lambda_{min} \tag{5.135}$$

$$\delta x = \frac{\alpha}{k_{xmax}} \tag{5.136}$$

$$\delta x = \frac{\alpha v}{2f_{max}\sin(\theta_{max})} = \frac{\alpha}{2\sin(\theta_{max})}\lambda_{min} \tag{5.137}$$

$$\delta x \sim \frac{1.4}{\sin(\theta_{max})}\lambda_{min} = \frac{1.4v}{f_{max}\sin(\theta_{max})} = \frac{vT}{2} \tag{5.138}$$

In this expression, k_{xmax} is the limiting wavenumber to be expected from a migration with no limitation on scattering angle. In any practical setting, there is always a limit on scattering angle and it is generally spatially variant. The limit may be a result of the effects of finite aperture, finite record length, and discrete spatial sample size or any of many other possibilities including: the migration algorithm may be "dip limited", lateral or complex vertical velocity variations can create shadow zones, attenuation effects are dependent on raypath length and hence affect the larger scattering angles more. Whatever the cause, a limitation of the range of scattering angles which can be collected and focused to a particular point translates directly into a resolution limit as expressed by equation (1). The size of the smallest resolvable feature, say δx, is inversely proportional to k_{xlim}. For definiteness, let $k_{xlim} = \alpha/(2\delta x)$, where α is a proportionality constant near unity, and solve for δx to get

$$\delta x = \frac{\alpha v}{4f_{max}\sin(\theta_{max})}. \tag{5.139}$$

Since the aperture limit is x and z variant (record length and spatial aliasing limits are z variant) δx must also vary with position. An interpretation of equation (5.139) is that it gives the smallest discernible feature on a reflector whose dip is normal to the bisector of the scattering angle cone at any position (Figure 5.71).

For constant velocity, the limits imposed on zero-offset scattering angle are easily derived using straight ray theory and have been shown (Lynn and Deregowski, 1981) to be

$$\tan\theta_A = \frac{A}{z} \tag{5.140}$$

and

$$\cos\theta_T = \frac{vT}{2z}. \tag{5.141}$$

In these expressions, A is the available aperture, T is the record length, z is depth, and v is the presumed constant velocity. θ_A and θ_T are limitations on scattering angle imposed by aperture and record length respectively (Figure 5.71). Aperture is defined as the horizontal distance from an analysis point to the end of the seismic line or the edge of a 3-D patch and is thus dependent on

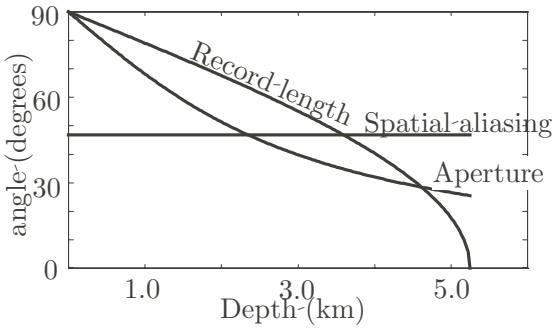

Figure 5.73: Constant velocity scattering angle chart showing the aperture, record length, and spatial aliasing limits for $A = 2500$m, $T = 3.0$sec, $v = 3500$m/s, $\Delta x = 20$m and $f = 60$Hz.

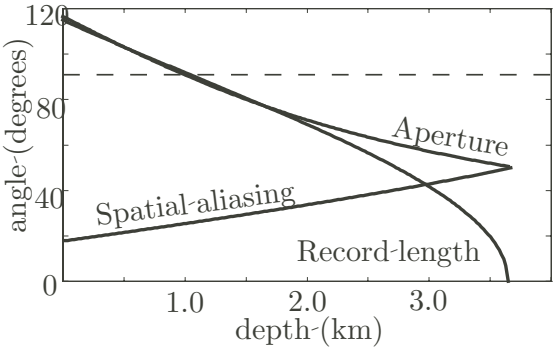

Figure 5.74: Constant gradient scattering angle chart showing the aperture, record length, and spatial aliasing limits for $A = 2500$m, $T = 3.0$sec, $v = 1500 + .6z$m/s, $\Delta x = 20$m and $f = 60$Hz.

azimuth and position (Figure 5.72). Alternatively, the record length limit has no lateral variation. Taken together, these expressions limit the scattering angle spectrum to a recordable subset.

A third limiting factor is spatial aliasing which further constrains the possible scattering angle spectrum to that which can be properly imaged (migrated). (Liner and Gobeli (1996) and Liner and Gobeli (1997) give an analysis of spatial aliasing in this context.) The Nyquist requirement is that there must be at least two samples per horizontal wavelength to avoid aliasing

$$\lambda_x = \frac{\lambda}{sin\theta_x} \geq 2\Delta x. \tag{5.142}$$

Here, Δx is the spatial sample size (CMP interval), λ and λ_x are wavelength and its apparent horizontal component, and θ_x is most properly interpreted as the emergence angle of a dipping event on a zero offset section. Consistent with zero-offset migration theory, the exploding reflector model Lowenthal and Sherwood (1976) can be used to relate wavelength to velocity through $\lambda = v/(2f)$ where f is some frequency of interest. This leads to an angle limited by spatial aliasing given by

$$\sin\theta_x = \frac{v}{4f\Delta x} \tag{5.143}$$

In the constant velocity case, the emergence angle of a ray and the scattering angle at depth are equal and thus equation (5.143) expresses the constant velocity limit on scattering angle imposed by spatial aliasing. For vertical velocity variation, $v(z)$, the result still applies provided that θ_x is simply interpreted as emergence angle and v as near surface velocity. The emergence angle can be related to the scattering angle at depth using Snell's law. This is done by recalling that the ray parameter, $p = sin(\theta(z))/v(z)$, is conserved (Slotnick, 1959), which leads to

$$\sin\theta_x = \frac{v(z)}{4f\Delta x}. \tag{5.144}$$

This expression generalizes spatial aliasing considerations to monotonically increasing but otherwise arbitrary $v(z)$ and θ_x is interpreted as the scattering angle from depth, z.

Returning to constant velocity, equations (5.140), (5.141), and (5.143) can be used to create a scattering angle resolution chart for an assumed recording geometry, position on the line, frequency of interest and constant velocity. A typical case is shown in Figure 5.73 where it is seen that the aperture limit is concave upward and tends asymptotically to zero at infinite depth. The record length limit has the opposite curvature and reaches zero degrees at the depth $z = vT/2$. Both limits admit the possibility of 90° only for $z = 0$. The spatial aliasing limit is depth independent but requires a frequency of interest which can conservatively be taken as the maximum (not dominant) signal frequency.

Charts such as Figure 5.73 can be used as an aid in survey design but tend to give unrealistic parameter estimates due to the assumption of straight raypaths. In most exploration settings, velocity increases systematically with depth and thus raypaths bend upward as they propagate from the scatterpoint to the surface. Intuitively, this should lead to shorter aperture requirements and allow the possibility of recording scattering angles beyond 90° in the near surface. The spatial aliasing limit has already been discussed in this context and the aperture and record length limits will now be derived exactly for the case of a constant velocity gradient, that is when $v(z) = vo + cz$. The derivation requires solution of the Snell's law raypath integrals for the linear gradient case (Slotnick, 1959). If p_A is the ray parameter required to trace a ray from a scatterpoint to the end of the spatial aperture, then

$$A(z) = \int_0^z \frac{p_A v(z')}{\sqrt{1 - p_A^2 v(z')^2}} dz'. \tag{5.145}$$

Similarly, let p_T be the ray parameter for that ray from scatterpoint to the surface which has traveltime (two-way) equal to the seismic record length, then

$$T(z) = \int_0^z \frac{1}{v(z')\sqrt{1 - p_T^2 v(z')^2}} dz'. \tag{5.146}$$

These integrals can be computed exactly, letting $v(z) = vo + cz$, to give

$$A = \frac{1}{p_A c}\left[\sqrt{1 - p_A^2 v_0^2} - \sqrt{1 - p_A^2 v(z)^2}\right], \tag{5.147}$$

and

$$T = \frac{2}{c}\ln\left[\frac{v(z)}{v_0}\left\{\frac{1 + \sqrt{1 - p_T^2 v_0^2}}{1 + \sqrt{1 - p_T^2 v(z)^2}}\right\}\right]. \tag{5.148}$$

Equations (5.147) and (5.148) give spatial aperture, A, and seismic record length, T, as a function of ray parameter and velocity structure.

Letting $p_A = sin(\theta_A(z))/v(z)$ and $p_T = sin(\theta_T(z))/v(z)$, equations (5.147) and (5.148) can both be solved for scattering angle to give

$$sin^2(\theta_A) = \frac{[2Acv_0\gamma]^2}{[A^2c^2 + v_0^2]^2\gamma^4 + 2v_0^2\gamma^2[A^2c^2 - v_0^2] + v_0^4}, \tag{5.149}$$

and

$$\cos(\theta_T) = \frac{\gamma^{-1} - \cosh(cT/2)}{\sinh(cT/2)}, \tag{5.150}$$

where

$$\gamma = \frac{v_0}{v(z)}. \tag{5.151}$$

When equations (5.144), (5.149), and (5.150) are used to create a scattering angle resolution chart, the result is typified by Figure 5.74. The parameters chosen are the same as for Figure 5.73 and the linear velocity function was designed such that it reaches 3500 m/s (the value used in Figure 5.73) in the middle of the depth range of Figure 5.74. It can be seen that the possibility of recording angles beyond 90° is predicted for the first 1000 m and the aperture limit is everywhere more broad than in Figure 5.73. The record length limit forces the scattering angle spectrum to zero at about 3700 m compared to over 5000 m in the constant velocity case. This more severe limit is not always the case, in fact a record length of 6 seconds will penetrate to over 12000 m in the linear velocity case and only 10500 m in the constant case. Also apparent is the fact that the spatial aliasing limit predicts quite severe aliasing in the shallow section though it gives exactly the same result at the depth where $v(z) = 3500$ m/s.

5.7.2 Examples

Figures 5.75A, 5.75B, and 5.75C provide further comparisons between the linear v(z) theory and constant velocity results. In Figure 5.75A, the aperture limits are contrasted for the same linear velocity function (v = 1500 +.6 z m/s) and constant velocity (v 3500 m/s) used before. The dark curves show the linear velocity results and the light curves emanating from 90° at zero depth are the constant velocity results. Each set of curves covers the range of aperture values (from top to bottom): 1000, 4000, 12000, and 20000 meters. The dramatic effect of the v(z) theory is especially obvious for larger apertures which admit angles beyond 90° for a considerable range of depths. Figure 5.75B is similar to Figure 5.75A except that the record length limit is explored. For each set of curves, the record lengths shown are (from top to bottom): 2.0, 4.0, 6.0, and 8.0 seconds. Figure 5.75C shows spatial aliasing limits for a frequency of 60 Hz. and a range of Δx values (from top to bottom): 10 20 40 60 80 100 and 150 meters.

Next consider Figure 5.76 that shows a synthetic demonstration of the aperture effect. Here, a number of unaliased point diffractor responses have been arranged at constant time. Thus the record length limit is constant and the aliasing limit does not apply. When migrated, the resulting display clearly shows the effect of finite spatial aperture on resolution. Comparison with Figure 5.72 shows the direct link between recorded scattering angle spectrum and resolution. Approximately, the focal points appear as dipping reflector segments oriented such that the normal (to the segment) bisects the captured scattering angle spectrum.

Figure 5.77 is a study designed to isolate the effects of temporal record length on resolution. The unmigrated section is constructed such that all five point diffractors are limited by record length and not by any other effect. Upon migration, the focal points are all symmetric about a vertical axis but show systematic loss of lateral resolution with increasing time. As shown in Figures 5.73, 5.74, and 5.75B, the record length limit always forces the scattering angle spectrum to zero at the bottom of the seismic section. Equation (5.139) then results in a lateral resolution size that approaches infinity. This effect accounts for the often seen data 'smearing' at the very bottom of seismic sections.

Figure 5.78 shows the migration of a single diffraction hyperbola with three different spatial sample intervals to illustrate the resolution degradation that accompanies spatial aliasing. In Figure 5.78B, the migration was performed with a spatial sample rate of 4.5 m which represents a comfortably unaliased situation. In Figures 5.78C and 5.78D the sample intervals are 9 m (slightly aliased) and 18 m (badly aliased). The slightly aliased situation has not overly compromised resolution but the badly aliased image is highly degraded. Note that the vertical size of the image is unaffected.

In Figure 5.79, the effect of maximum temporal frequency is examined. A single diffraction hyperbola was migrated with three different maximum frequency limits . In Figure 5.79B, the focal point and its local (k_x, f) spectrum are shown for an 80 Hz. maximum frequency while Figures 5.79C

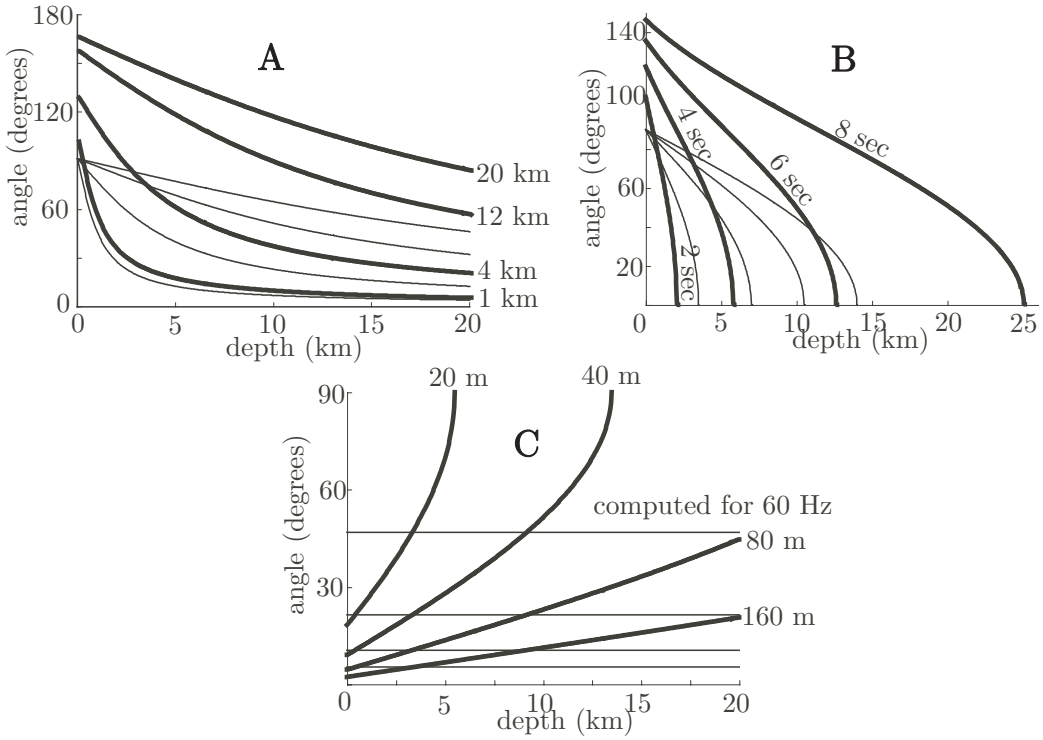

Figure 5.75: The constant and linear $v(z)$ theories are compared. In each case, the linear $v(z)$ theory is presented with bold lines while the constant velocity theory uses thin lines. The constant velocity used was 3500 m/s and the linear function was $v = 1500 + .6z$ m/s. (A) The aperture limits are contrasted. The apertures used are labeled on the bold curves but the light curves use sequentially the same A values. (B) The record limits are contrasted. The record lengths used are labeled on the bold curves but the light curves use sequentially the same T values. (C) The spatial aliasing limits are contrasted. The Δx values used are labeled on the bold curves but the light curves use sequentially the same values.

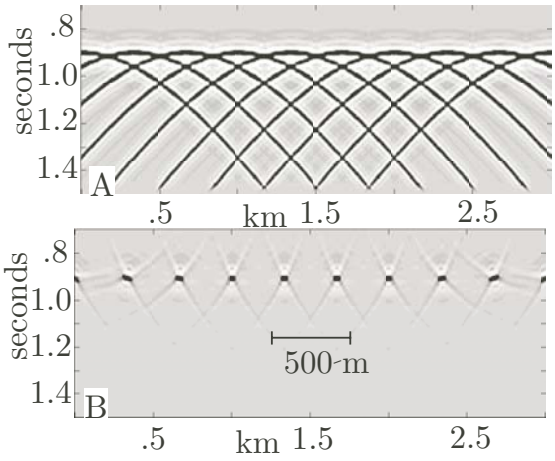

Figure 5.76: (A) A series of point scatterers all at the same time so that their record length effect is constant. (B) The migration of (A) shows the spatial variability of focal point geometry caused by the aperture effect. Compare with Figure 5.72

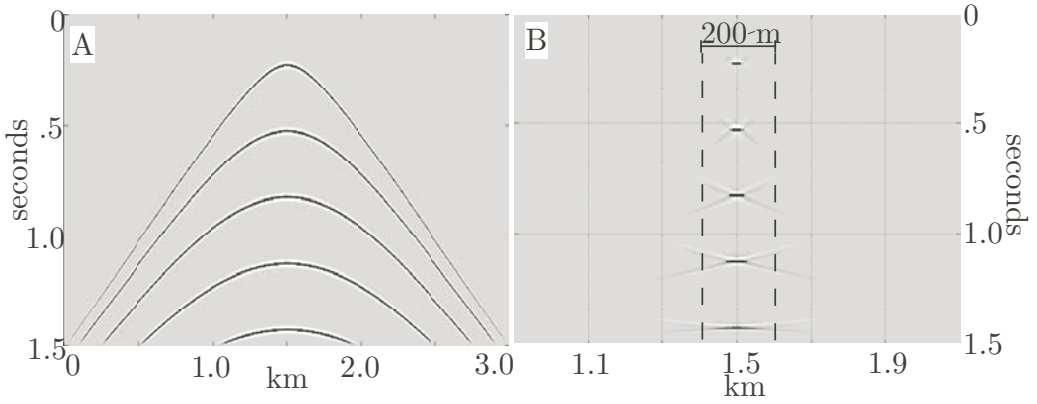

Figure 5.77: An illustration of the record length effect. (A) A series of diffractions that are all truncated by the bottom of the section to minimize the aperture effect. (B) The migration of (A) shows a systematic loss of lateral resolution with increasing time. At the bottom of the section, the lateral size of a focal point is theoretically infinite. Note the horizontal scale change between (A) and (B).

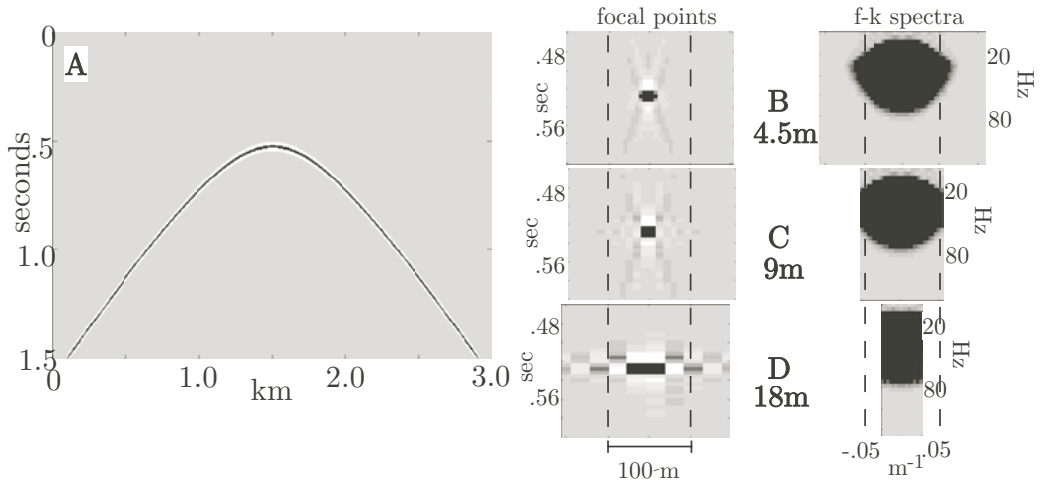

Figure 5.78: The spatial aliasing effect is demonstrated. (A) A single diffraction hyperbola. (B) A zoom of the focal point and the (k_x, f) spectrum after migration of (A) with $\Delta x =$ 4.5 m. (C) A zoom of the focal point and the (k_x, f) spectrum after migration of (A) with $\Delta x = 9$ m. (D) A zoom of the focal point and the (k_x, f) spectrum after migration of (A) with $\Delta x = 18$ m.

and 5.79D are similar except that the maximum frequencies were 60 Hz. and 40 Hz. respectively. It is clear from these figures that limiting temporal frequency affects both vertical and lateral resolution. As expected from equation (5.139), a reduction of f_{max} from 80 Hz to 40 Hz causes a doubling of the focal point size.

In summary, the theory of (k_x, f) migration predicts a simple model for the resolving power of seismic data. The result is a spatial bandwidth that depends directly on frequency and sine of scattering angle and inversely on velocity. Finite recording parameters (aperture and record length) place space and time variant limits on the observable scattering angle spectrum. Thus the resolution of a seismic line is a function of position within the aperture of the line. The scattering angle limits imposed by aperture, record length, and spatial sampling can be derived exactly for the case of constant velocity and for velocity linear with depth. The linear velocity results are more realistic and lead to considerably different, and usually cheaper, survey parameters than the constant velocity formulae.

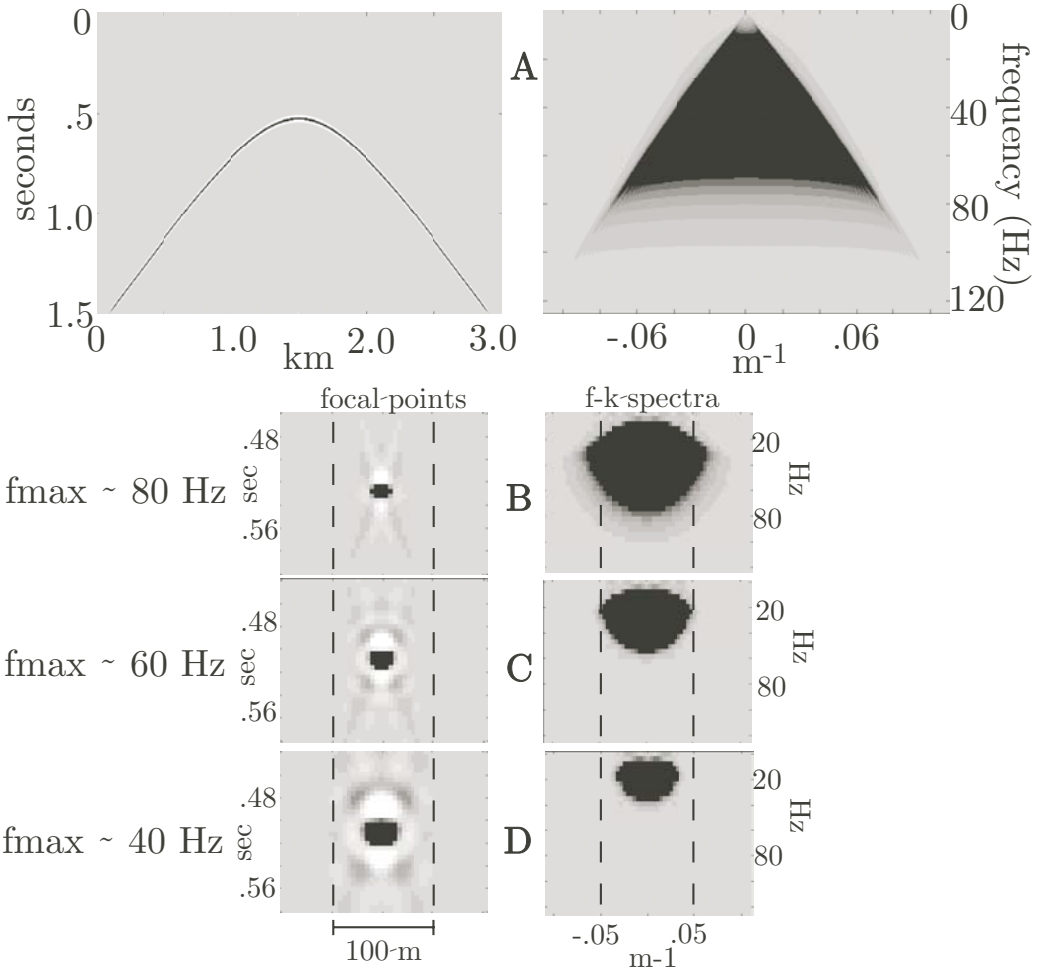

Figure 5.79: The resolution effect of decreasing F_{max} is demonstrated. (A) A single diffraction hyperbola and its (k_x, f) spectrum. (B) A zoom of the focal point and the (k_x, f) spectrum after migration of (A) with $f_{max} = 80$ Hz. (C) A zoom of the focal point and the (k_x, f) spectrum after migration of (A) with $f_{max} = 60$ Hz. (D) A zoom of the focal point and the (k_x, f) spectrum after migration of (A) with $f_{max} = 40$ Hz.

Chapter 6

Appendix A

There seems to be a little room here for brilliant expostulation.

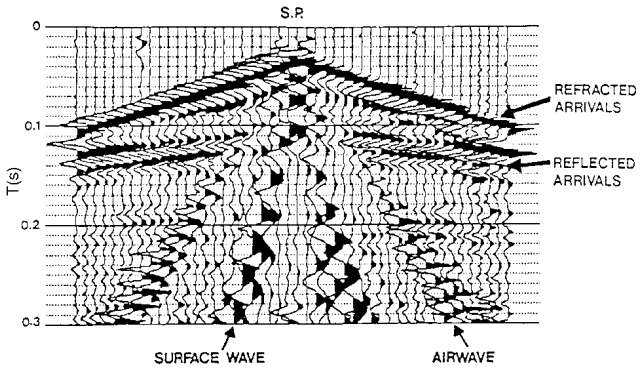

Figure 6.1: Seisgun field record showing reflections, refractions, surface waves and air blast (courtesy of Don Lawton).

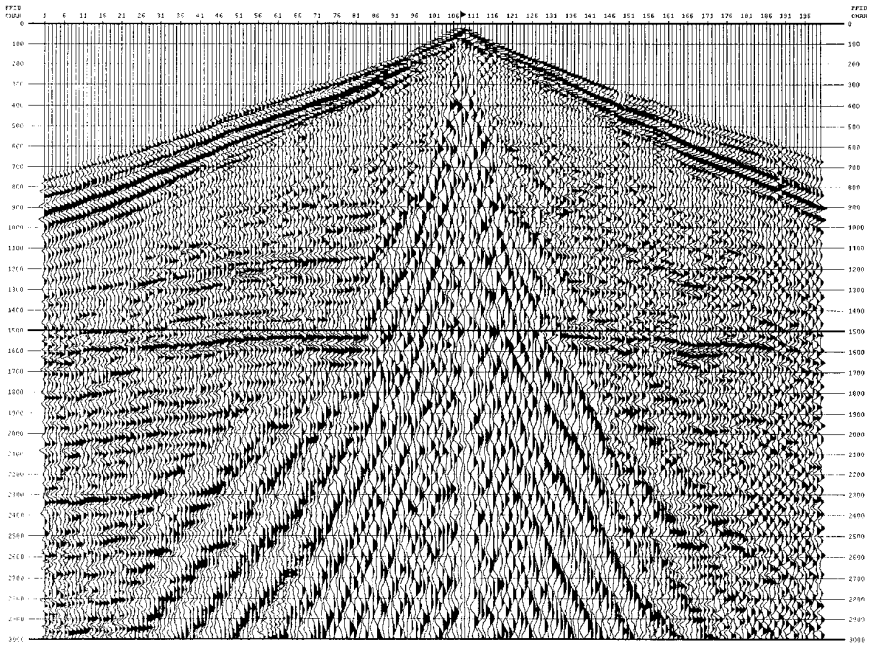

Figure 6.2: 2D Dynamite field record from Blackfoot, southern Alberta (courtesy of Don Lawton).

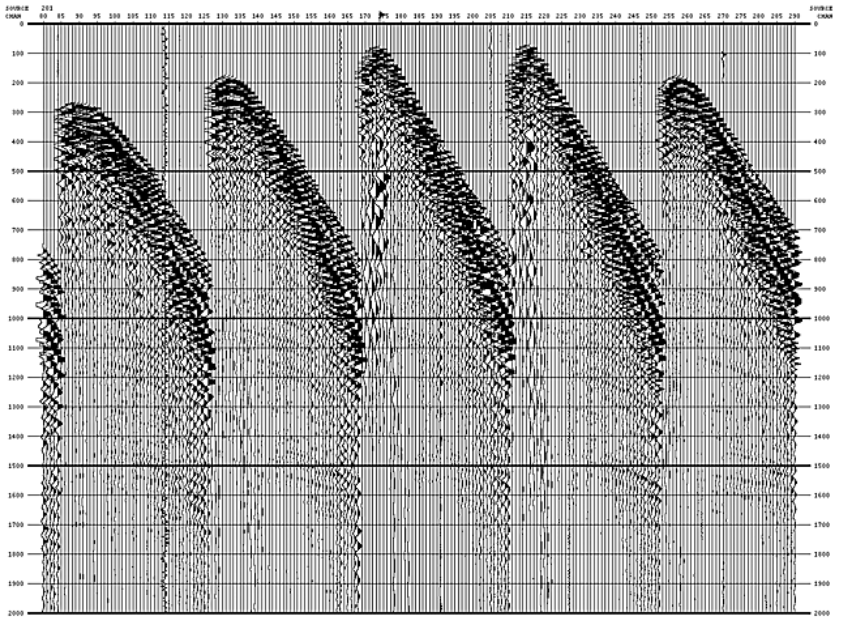

Figure 6.3: Part of a 3D dynamite field record from Blackfoot, southern Alberta (Simin et al., 1996).

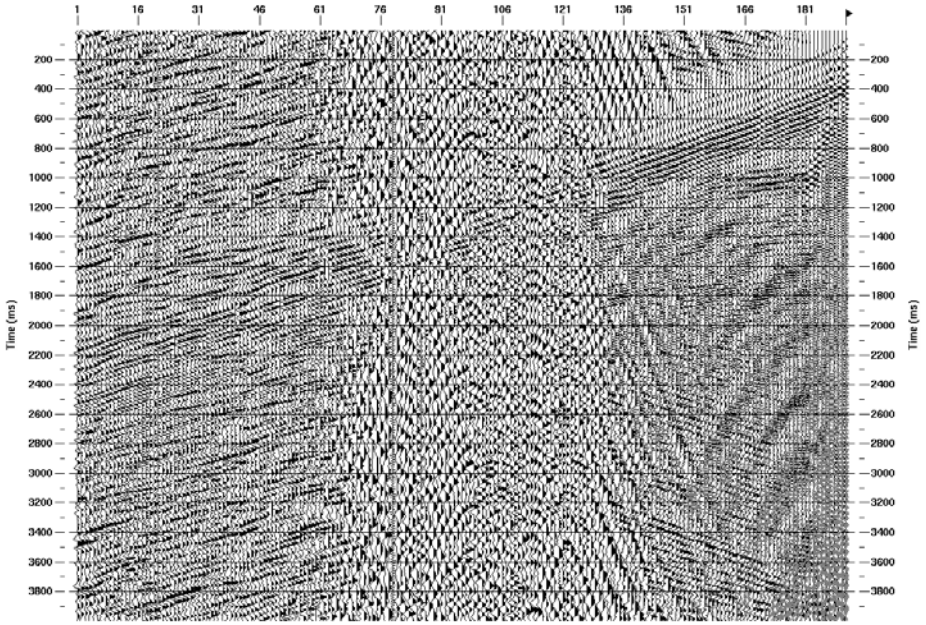

Figure 6.4: Uncorrelated VibroSeis Field record (with AGC) from Pikes Peak, Saskatchewan.

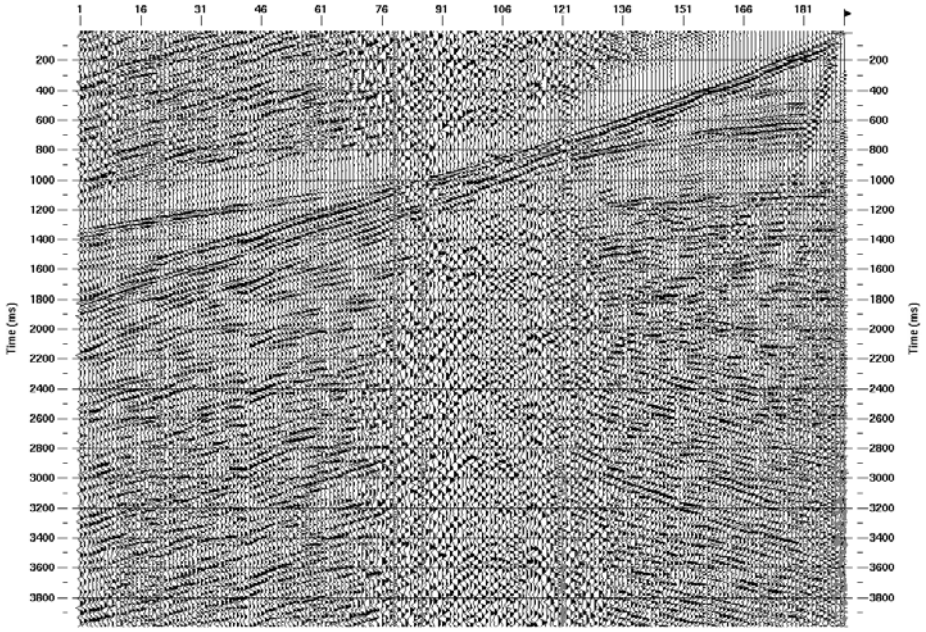

Figure 6.5: Same record as Figure 6.4, after correlation with the sweep, bandpass filter, and AGC.

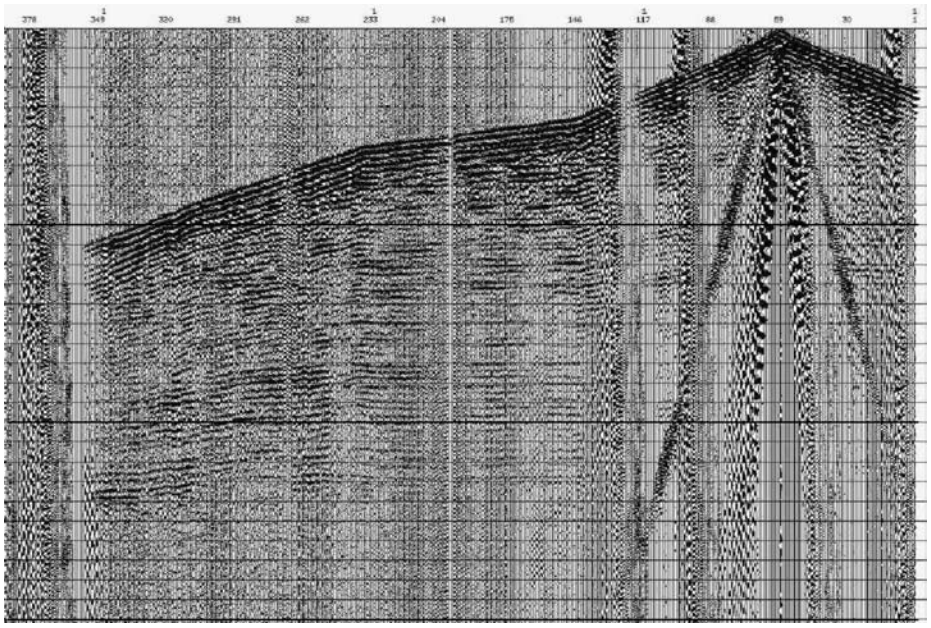

Figure 6.6: Correlated VibroSeis field record from Jumpingpound 3C-2D, after vertical stacking and AGC. Note surface noise, air blast, break in slope of direct arrivals, and road noise (Lawton et al., 2002).

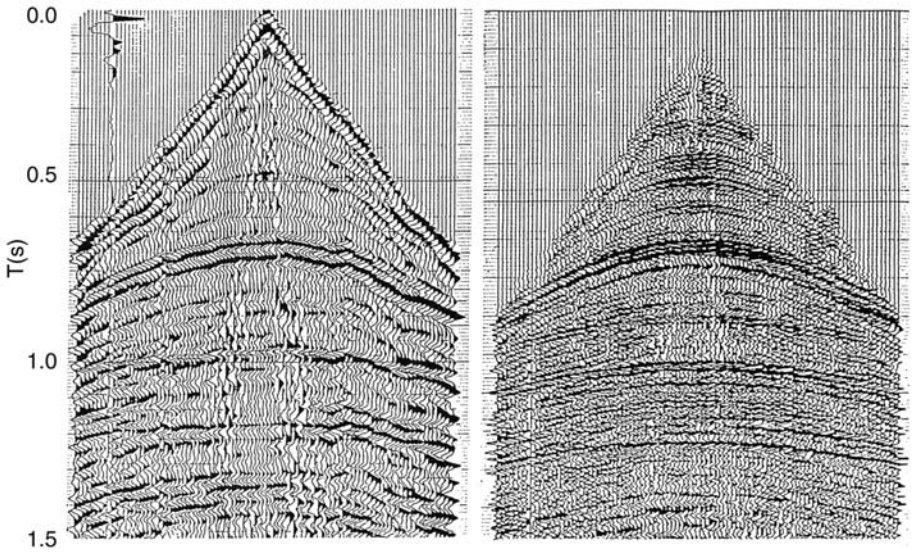

Figure 6.7: Left: Raw record. Right: After trace edit, top mute, statics, filtering, and deconvolution (courtesy of Don Lawton).

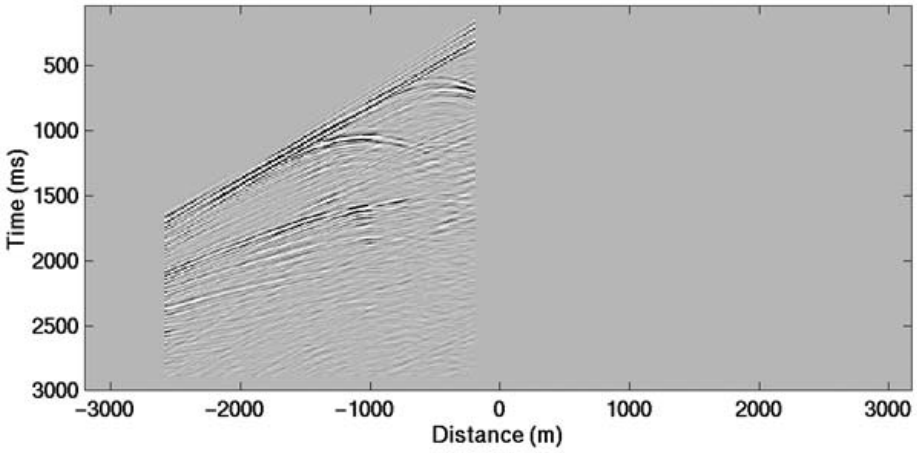

Figure 6.8: Unmigrated synthetic source gather from Marmousi model.

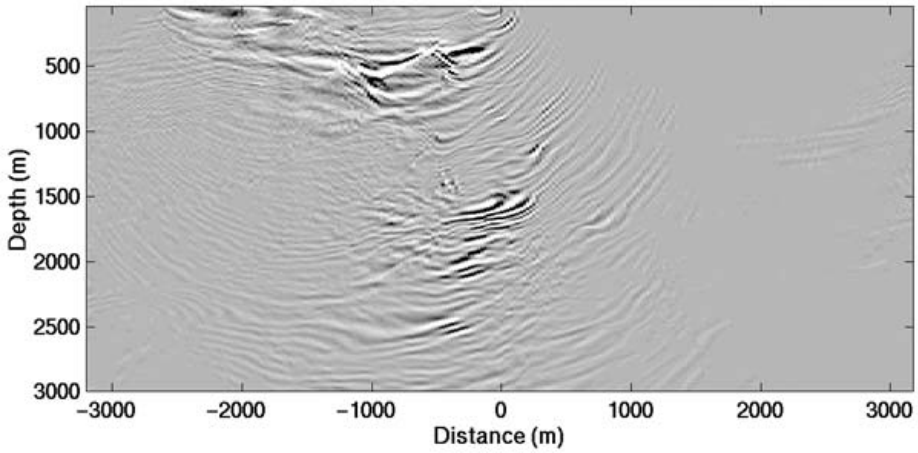

Figure 6.9: Same data as Figure 6.8, after pre-stack depth migration.

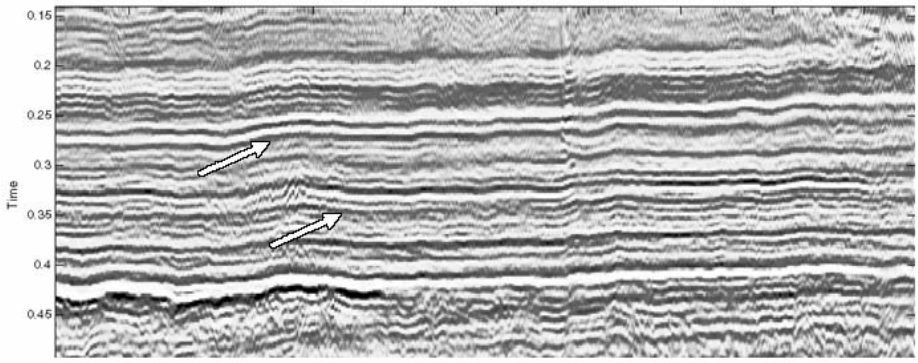

Figure 6.10: Blackfoot stack after surface consistent Wiener deconvolution and time-variant spectral whitening.

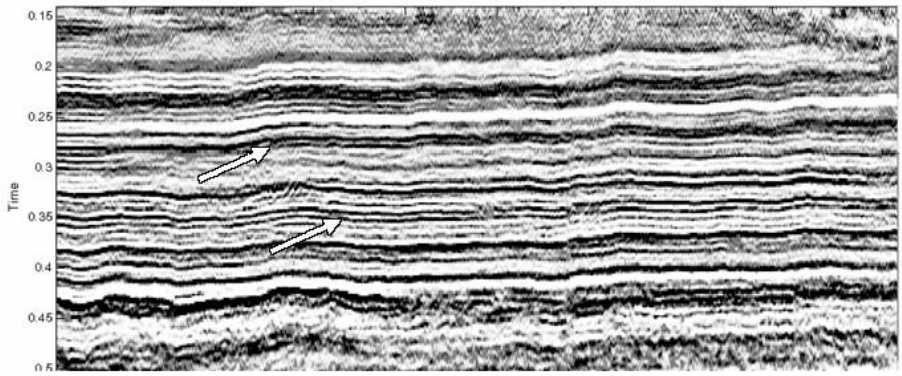

Figure 6.11: Blackfoot stack after Gabor deconvolution. Note improved resolution in time as compared to Figure 6.10.

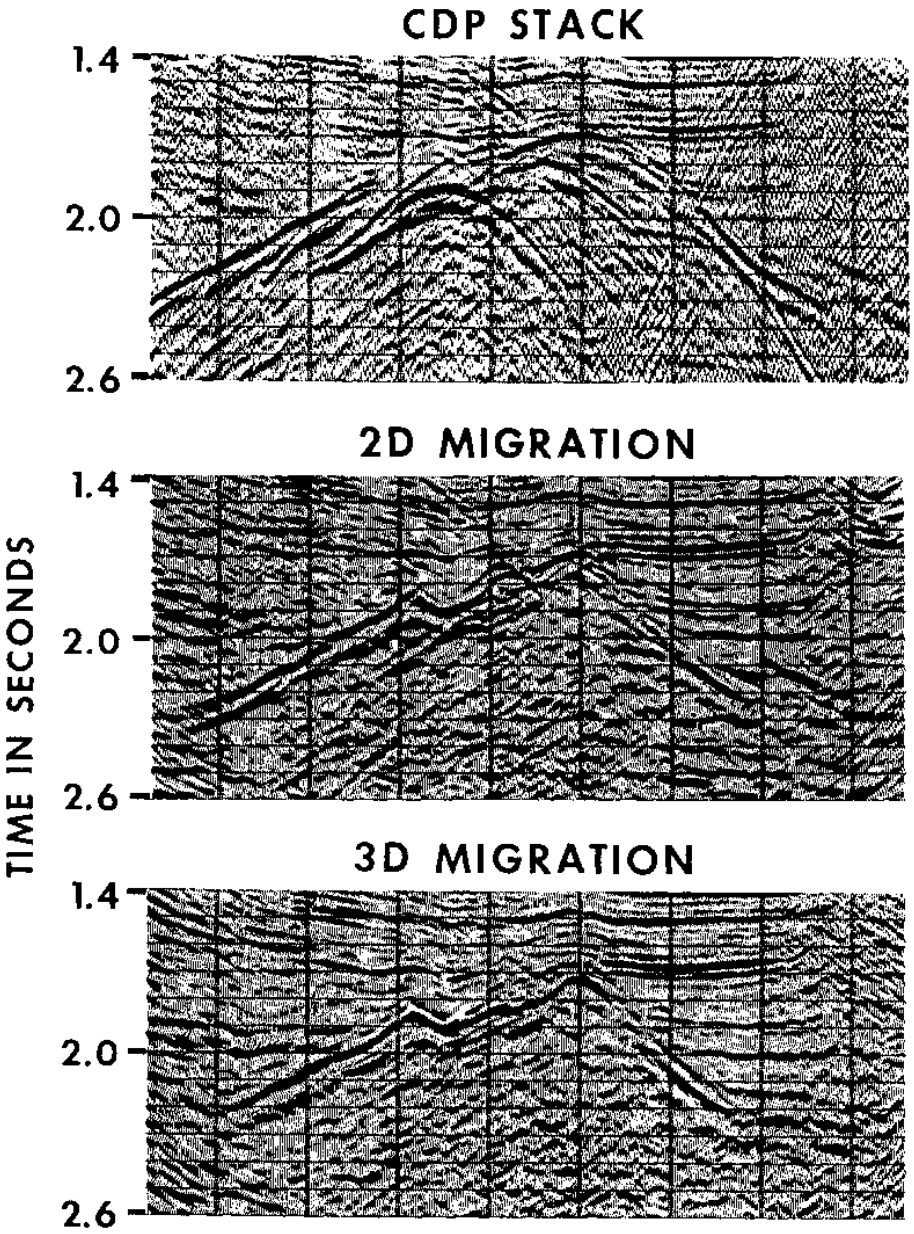

Figure 6.12: Top: CDP stack from 3D volume, Middle: after 2D migration, Bottom: after full 3D migration. The 2D migrated version contains out-of-plane reflections, resulting in a poorer image. (Brown, 1991).

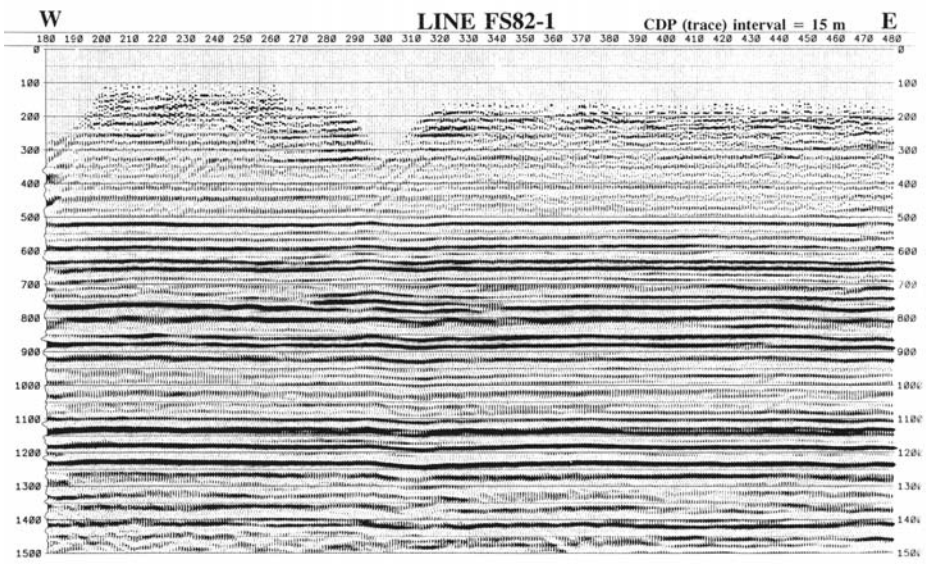

Figure 6.13: Typical data from Southern Alberta showing undeformed sedimentary basin strata. (courtesy of Don Lawton.)

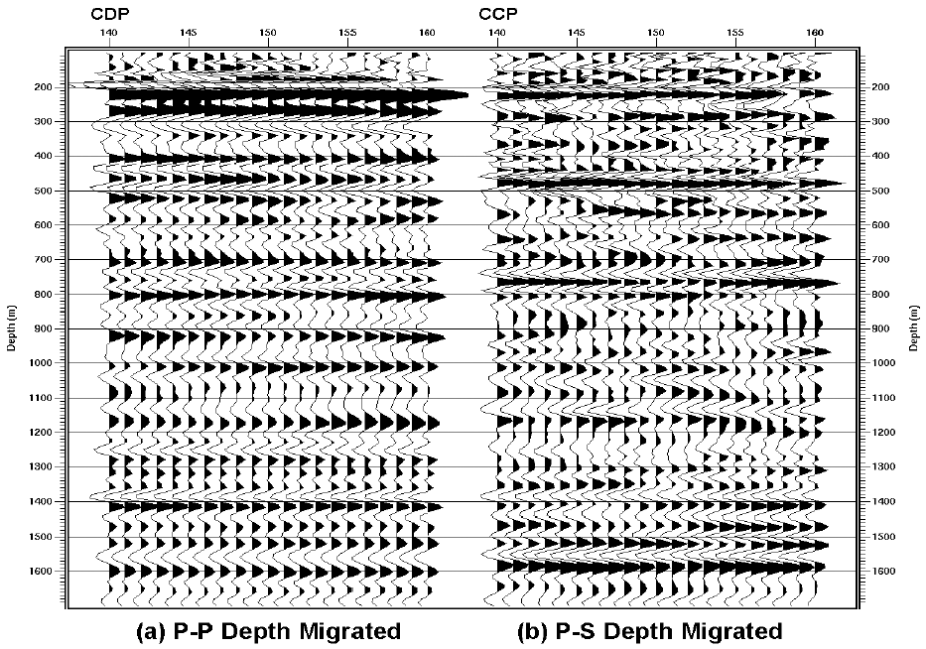

Figure 6.14: Comparison of depth migrated conventional and converted wave stacks from Blackfoot 3C-2D, Southern Alberta (Hoffe and Lines, 1998).

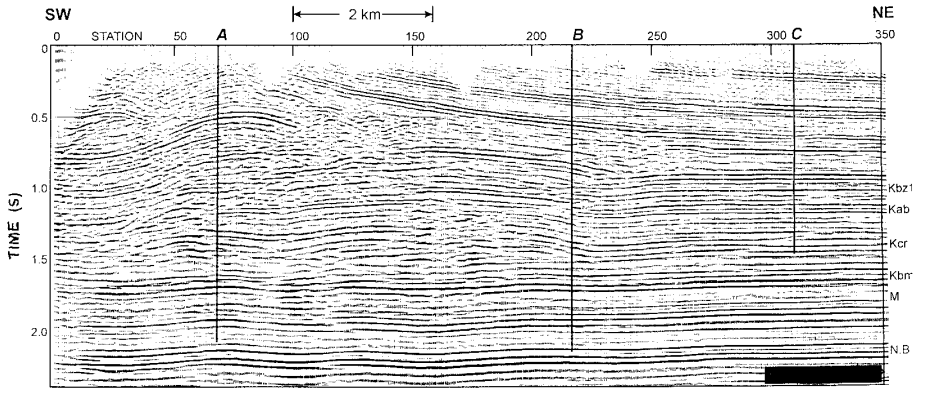

Figure 6.15: Triangle zone example from Southern Alberta. Sediments near the surface in the middle of the section have been pushed upwards above a back thrust, as a wedge of material was forced in from the southwest. (courtesy of Don Lawton).

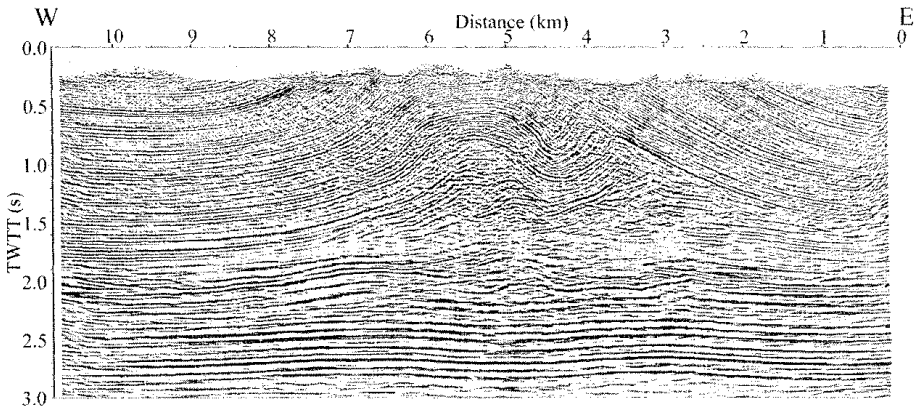

Figure 6.16: Another example of a triangle zone, Shaw Basing area (Yan and Lines, 2001).

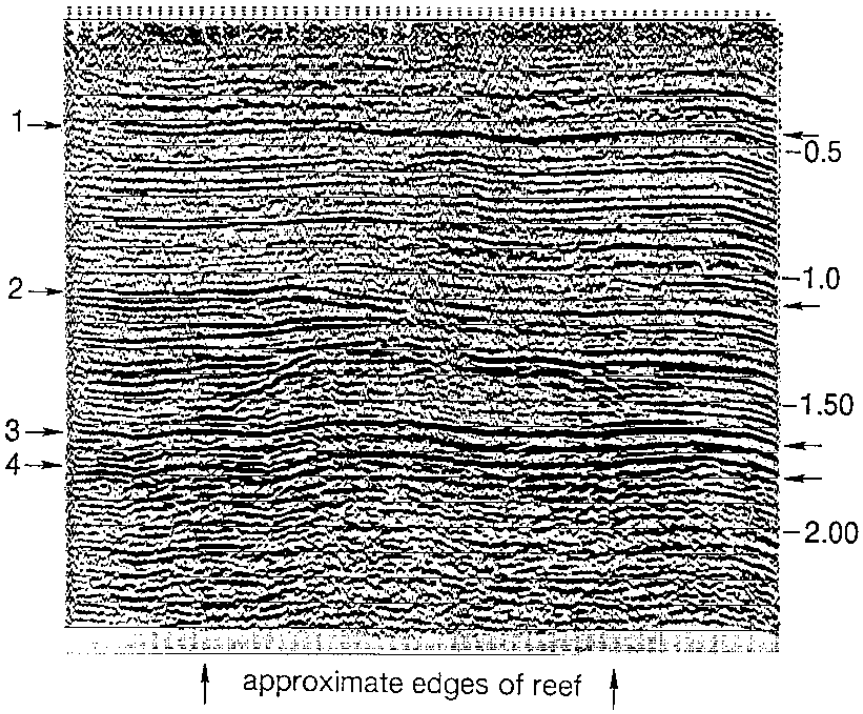

Figure 6.17: Unmigrated stack section showing a Permian carbonate reef in West Texas known as the Horseshoe Atoll (courtesy of Larry Lines).

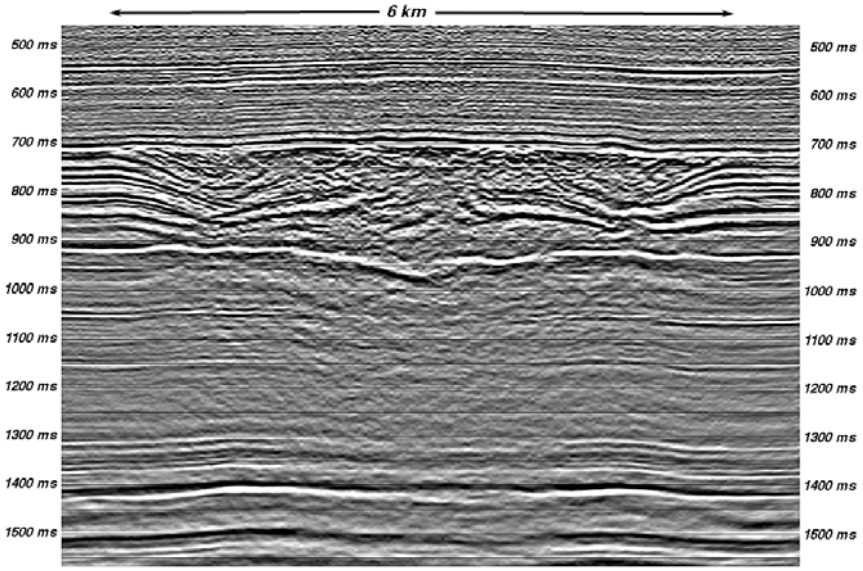

Figure 6.18: Hotchkiss structure, northern Alberta. This has been interpreted to be a (small) complex impact crater (Mazur and Stewart, 1997).

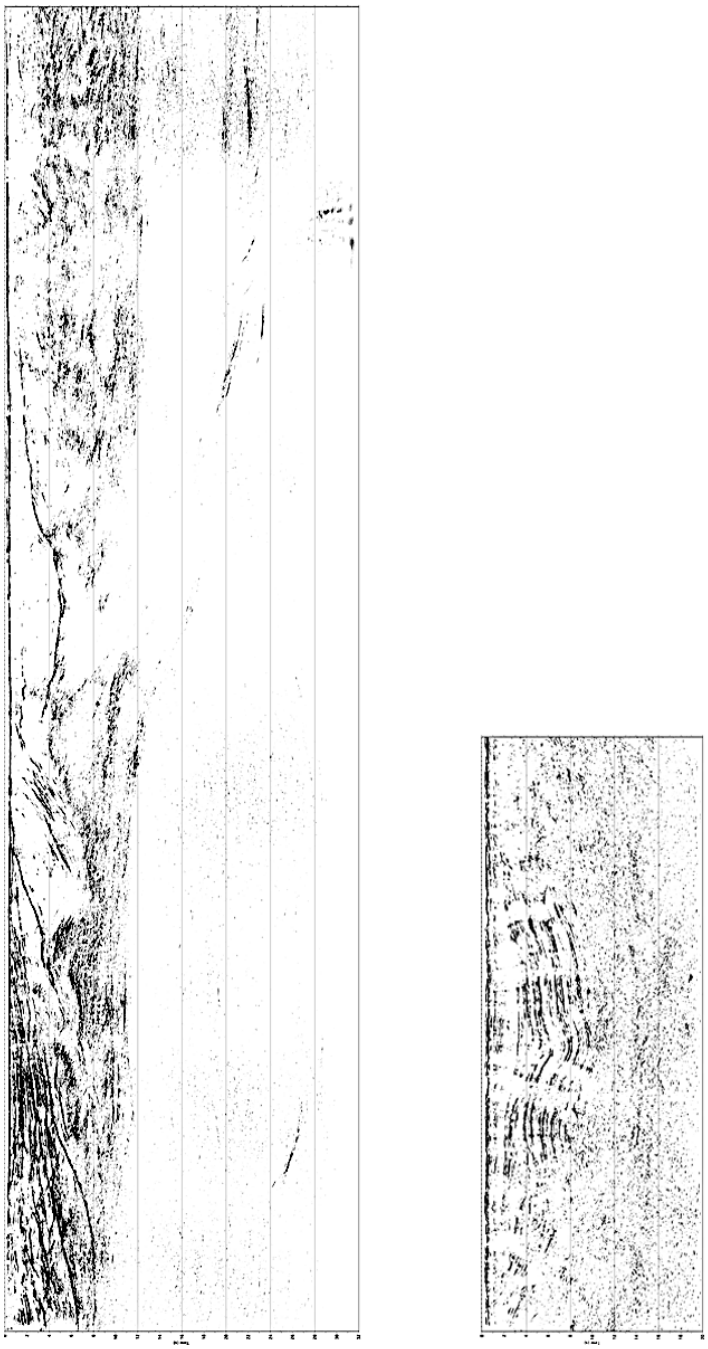

Figure 6.19: Left: Western half (no vertical exaggeration; width $\approx$ 116 km) of migrated and coherency filtered land seismic, west of Great Slave Lake, N.W.T. Right: Migrated and coherency filtered marine seismic from Lake Huron, Ontario, plotted at the same scale. (http:\\www.litho.ucalgary.ca\atlas\atlas.html).

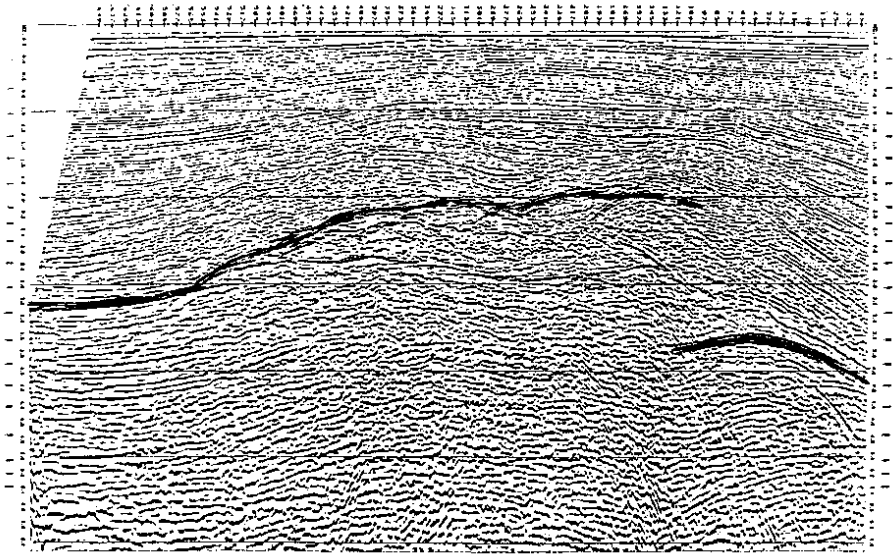

Figure 6.20: Unmigrated section from the Gulf of Mexico, with the top of a salt dome interpreted in black (Lines, 1991).

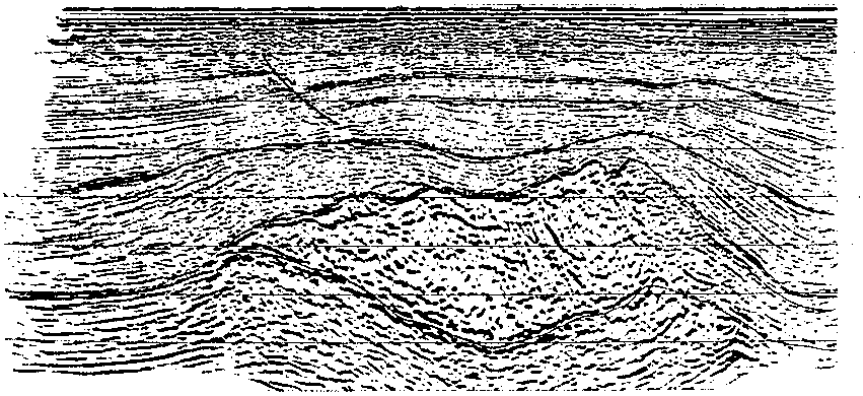

Figure 6.21: Same data as Figure 6.20. The salt dome is much more interpretable on a depth-migrated section (Lines, 1991).

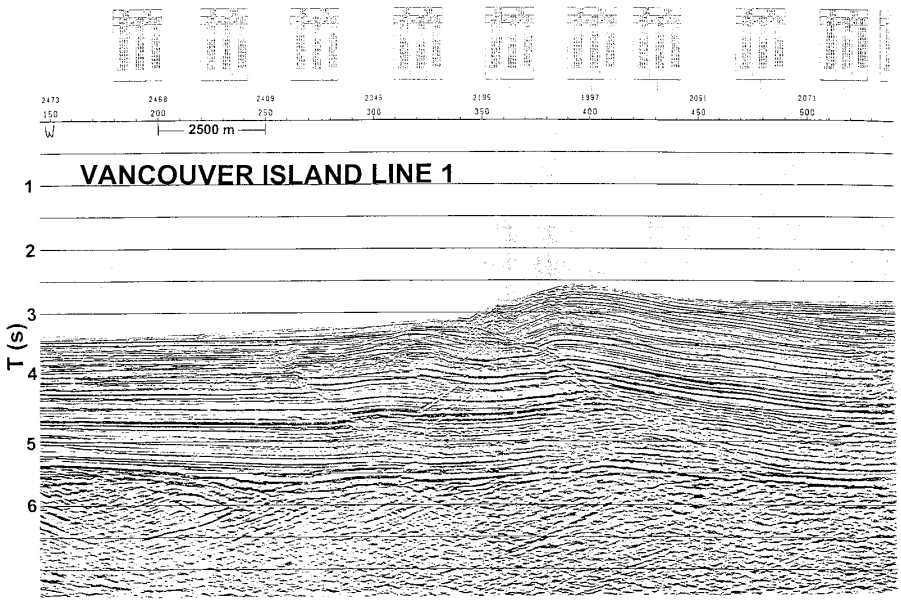

Figure 6.22: Deformation of marine sediments as a result of subduction of oceanic crust, west of Vancouver Island, B.C. (courtesy Pacific Geoscience Centre).

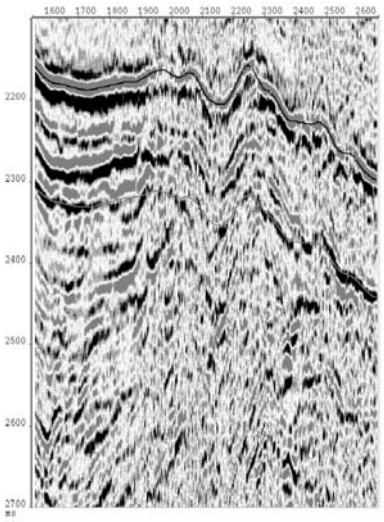

Figure 6.23: Offshore data from the North Sea with water-bottom and peg-leg multiples (Lokshtanov, 2002).

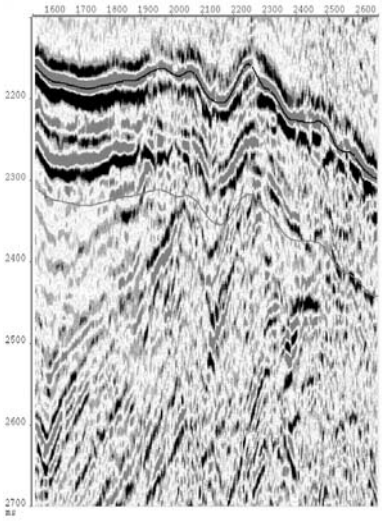

Figure 6.24: Same data as Figure 6.23 after wave-equation multiple suppression (Lokshtanov, 2002).

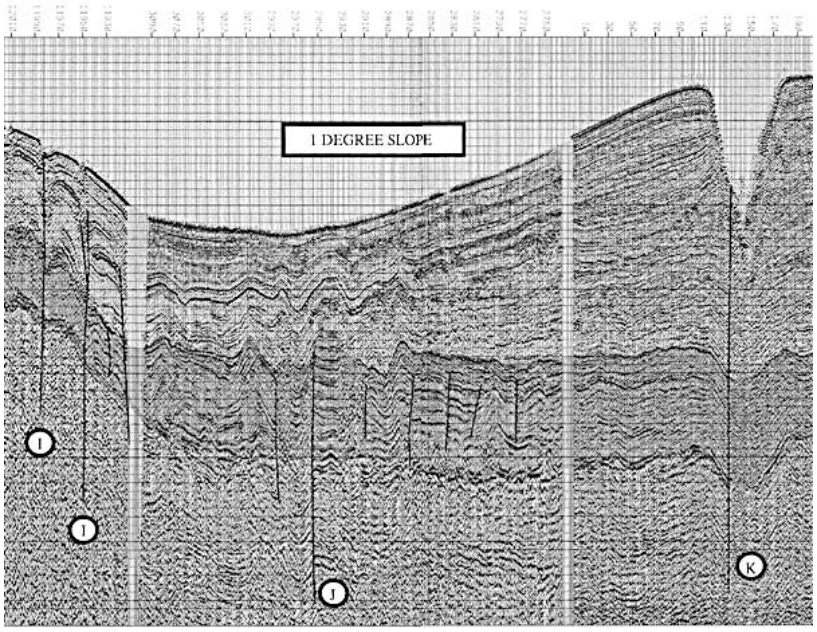

Figure 6.25: Portion of vertically exaggerated (1:30) strike line along the coast of Africa (courtesy of Amoco).

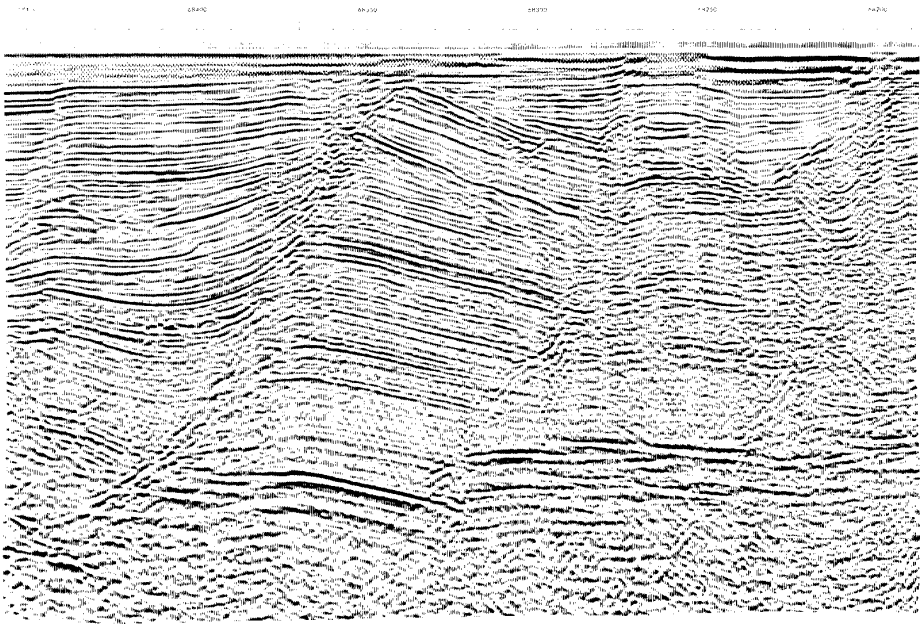

Figure 6.26: Growth faults, offshore New Zealand. Younger sediments continue to be deposited in basins created as the fault blocks drop down to the left (courtesy of Don Lawton).

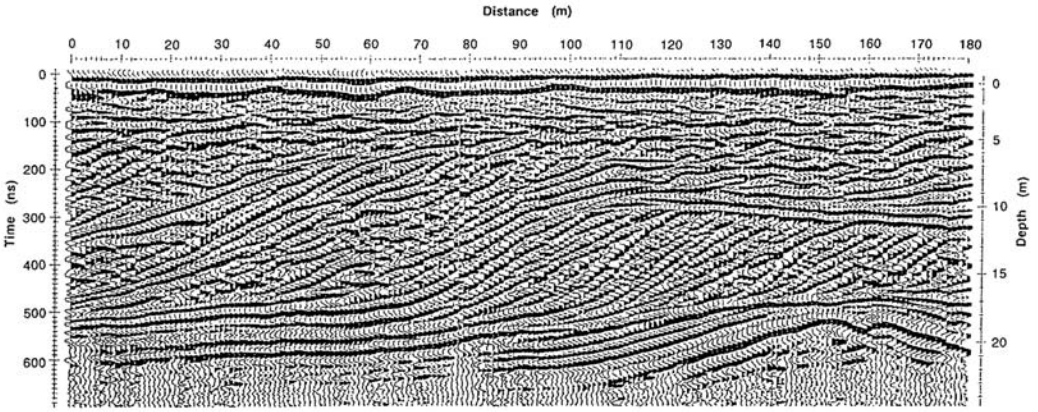

Figure 6.27: Ground penetrating radar (GPR) section in the William River Delta, Northern Saskatchewan (Jol and Smith, 1991).

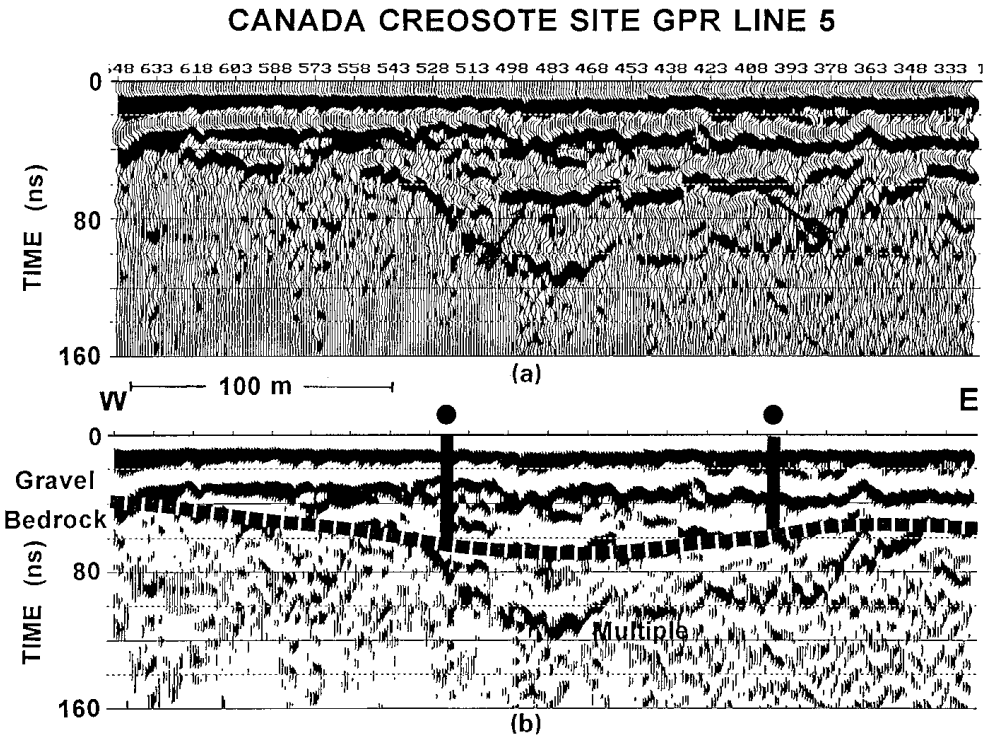

Figure 6.28: GPR section at a creosote contamination site on the Bow River, Calgary (Lawton and Jol, 1994).

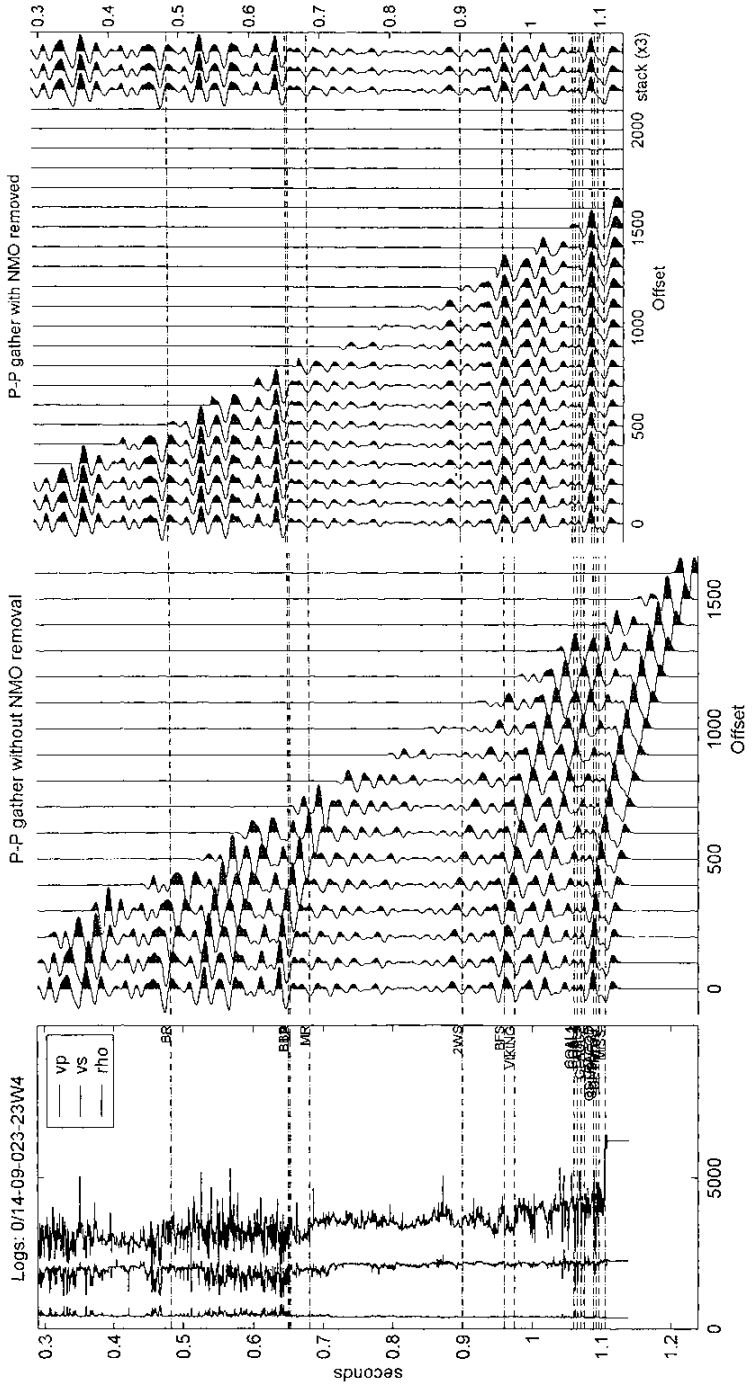

Figure 6.29: Left: Borehole sonic and density logs. Middle left: synthetic P-P gather (reflectivity convolved with wavelet) showing effect of Normal Moveout (NMO). Middle right: Same gather after NMO removal. Right: Synthetic trace after stacking (courtesy of Don Lawton).

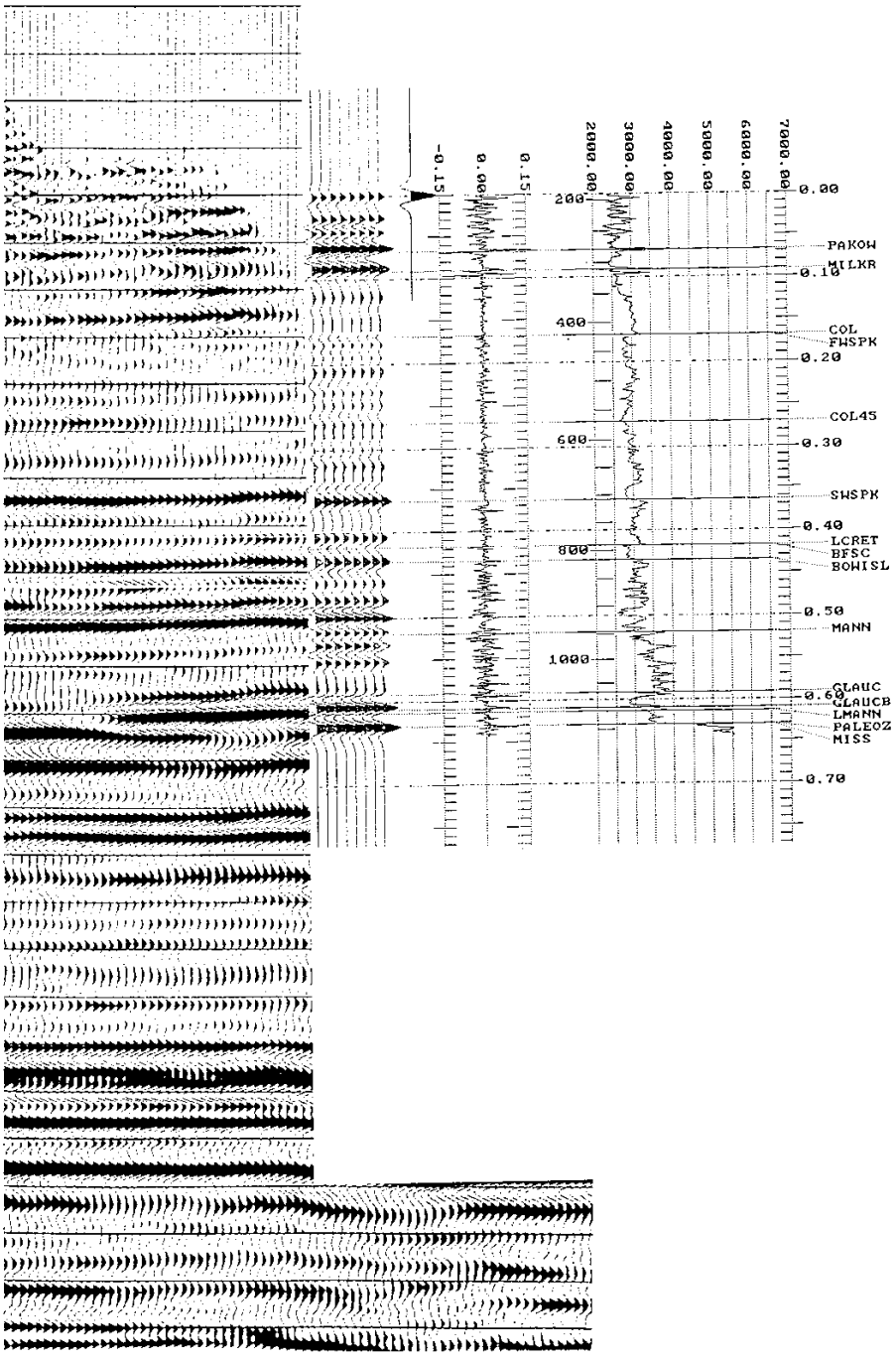

Figure 6.30: Left: CDP stack (same data as Figure 6.13). Middle: Synthetic seismogram and reflectivity. Right: Sonic logs used to calculate reflectivity (courtesy of Don Lawton).

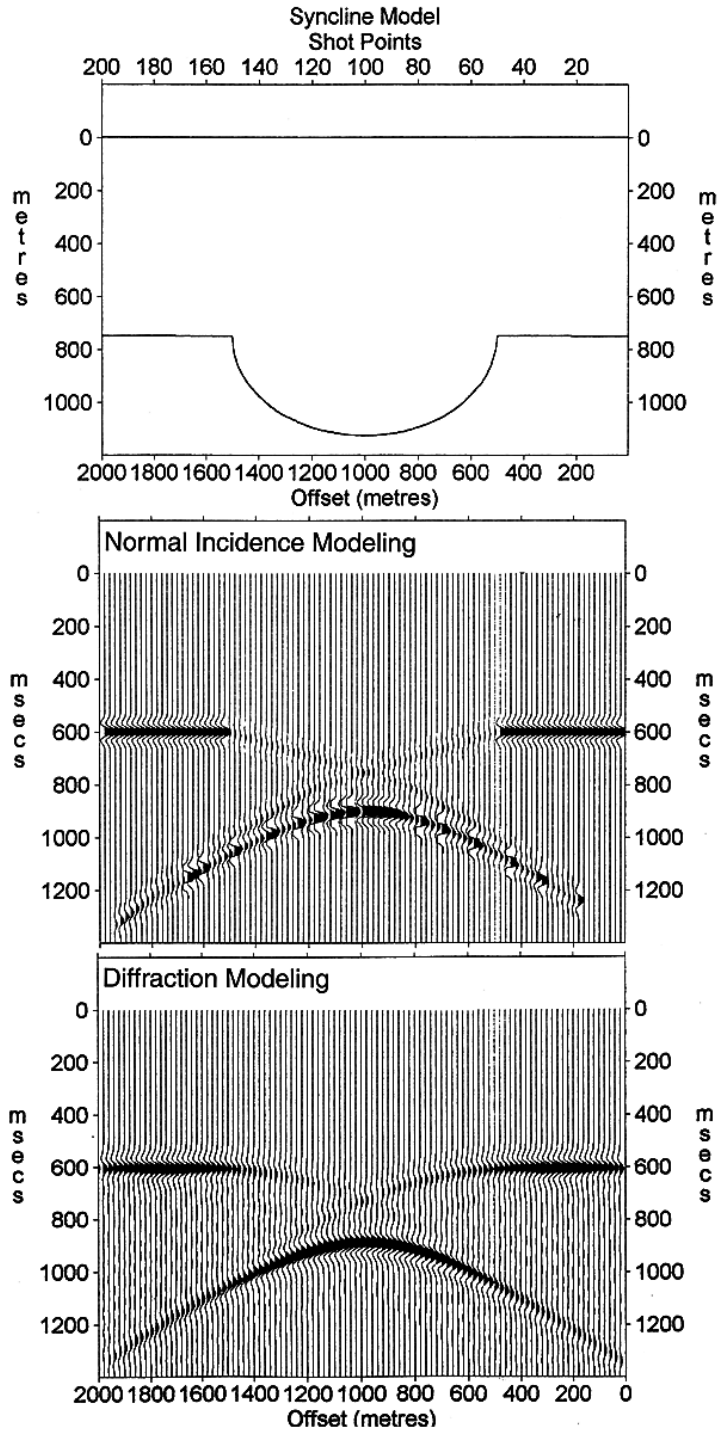

Figure 6.31: Comparison of normal incidence and diffraction modeling for a deep syncline (focal point below the surface, courtesy of Don Lawton).

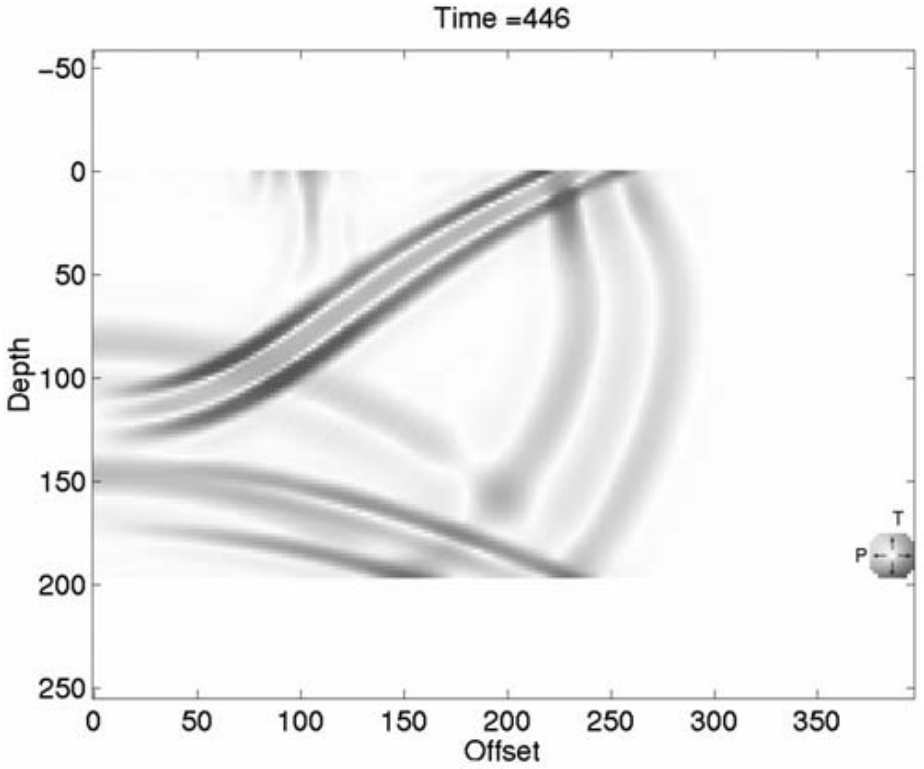

Figure 6.32: Finite difference snapshot showing compressional and shear waves moving away from a surface seismic source.

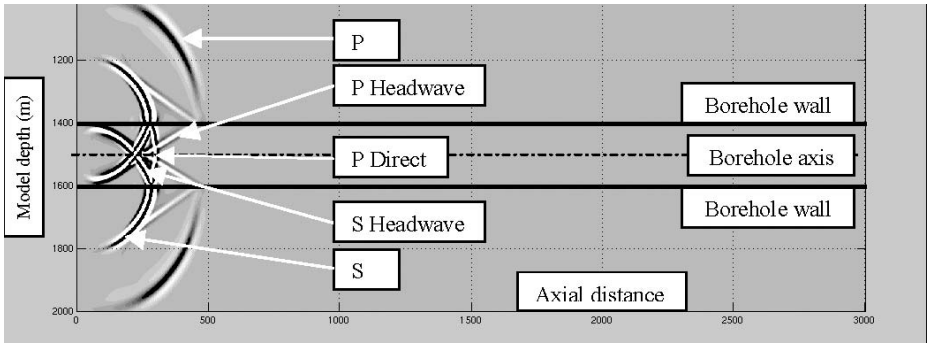

Figure 6.33: Finite difference snapshot showing compressional and shear waves moving away from a source within a fluid-filled borehole (Chabot et al., 2000).

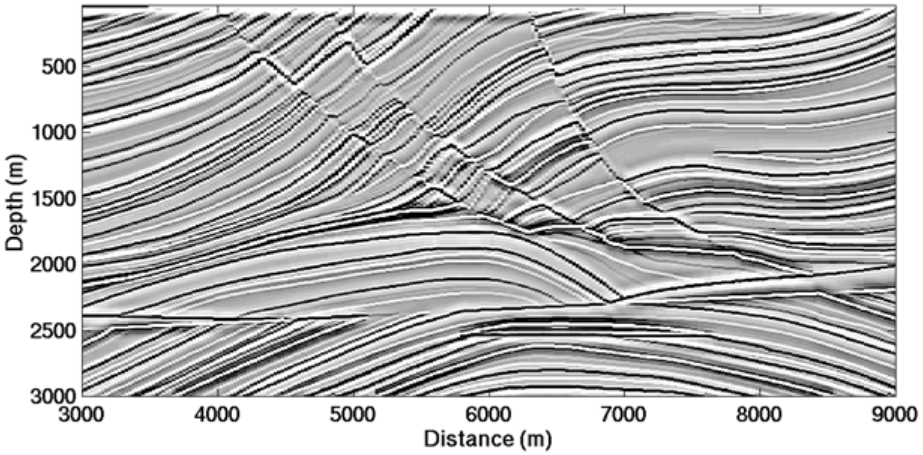

Figure 6.34: Marmousi model reflectivity. Synthetic shot gathers (eg. Figures 6.8 and 6.9) were produced from this model.

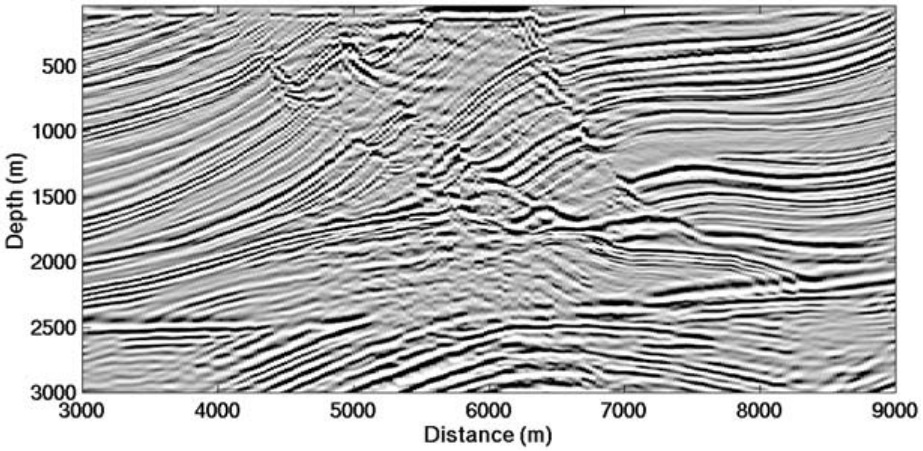

Figure 6.35: Marmousi stack after pre-stack depth migration. The goal is to produce an image that accurately represents the reflectivity shown in Figure 6.34.

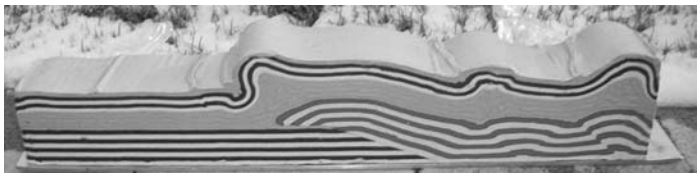

Figure 6.36: Silicone putty and plasticine fold-thrust model manufactured by C-CORE, Memorial University (Gallant et al., 2002).

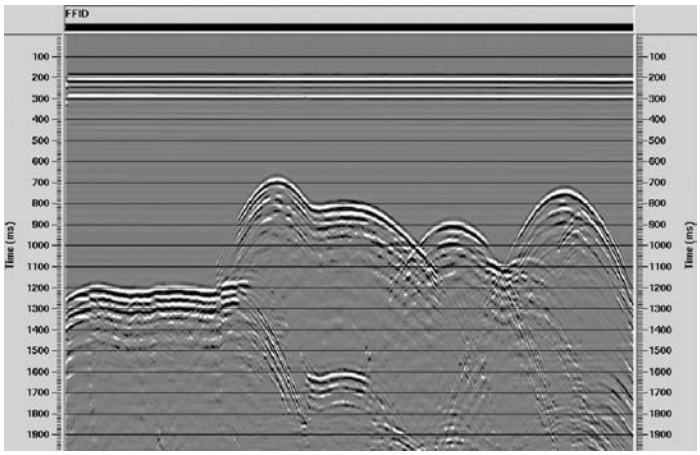

Figure 6.37: Zero-offset ultrasonic reflection data acquired over model shown in Figure 6.36 (Gallant et al., 2002).

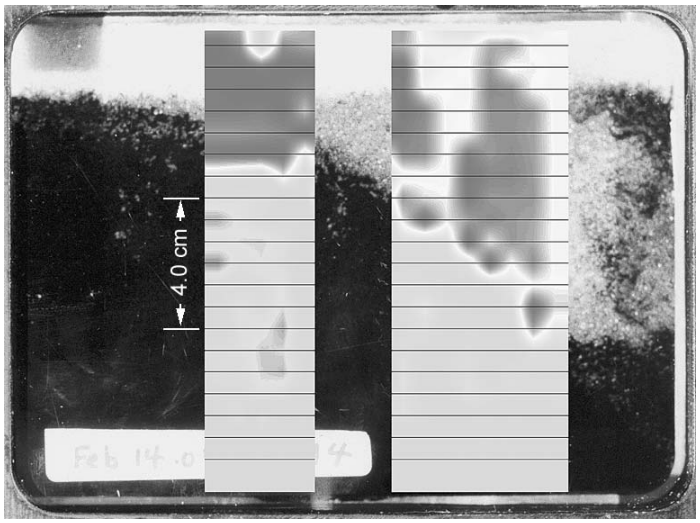

Figure 6.38: Time slice from 3D ultrasonic reflection survey over a heavy oil recovery model, superimposed on the physical model (Hall et al., 2001).

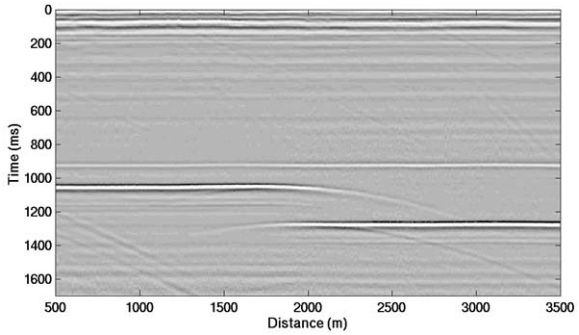

Figure 6.39: Ultrasonic data acquired over a model of a step below anisotropic overburden with a 45 degree dip.

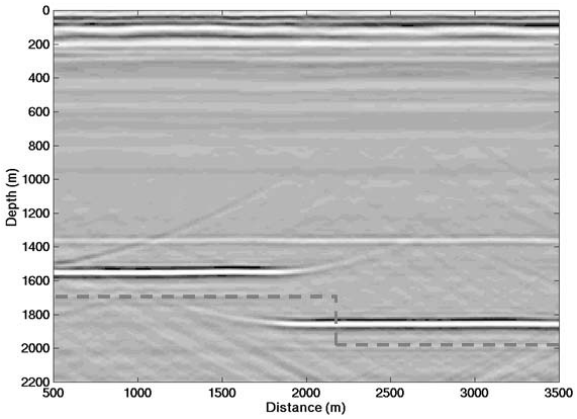

Figure 6.40: Figure 6.39 after isotropic depth migration. Note that the step is not positioned correctly.

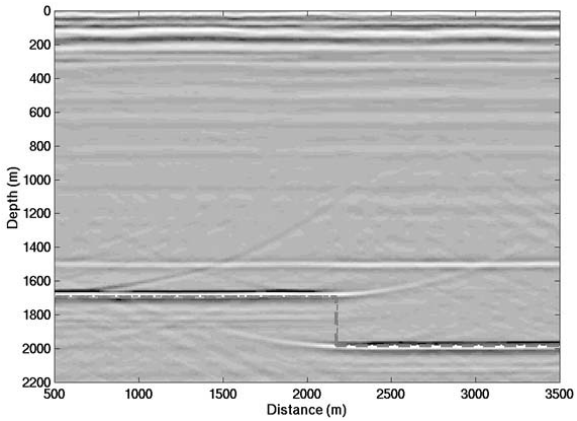

Figure 6.41: Figure 6.39 after anisotropic depth migration. The step is now positioned correctly.

Bibliography

Aki, K., and Richards, P. G., 1980, Quantitative Seismology: W.H. Freeman.

Ames, W. F., 1992, Numerical Methods for Partial Differential Equations: Academic Press.

Berkhout, A. J., 1985, Seismic Migration: Developments in solid earth geophysics:, volume 14 Elsevier.

Brown, A., 1991, Interpretation of three-dimensional seismic data:, volume 42 of **AAPG Memoir** AAPG, Tulsa, OK.

Chabot, L., Henley, D. C., and Brown, R. J., 2000, Single-well imaging using the full waveform of an acoustic sonic: CREWES Research Report, **12**.

Chun, J. H., and Jacewitz, C. A., 1981, Fundamentals of frequency domain migration: Geophysics, **46**, 717–733.

Claerbout, J. F., 1976, Fundamentals of Geophysical Data Processing: McGraw Hill.

Clayton, R., and Engquist, B., 1977, Absorbing boundary conditions for acoustic and elastic wave equations: Bull. Seis. Soc. Am., **67**, 1529–1540.

Cohen, L., 1995, Time Frequency Analysis: Prentice Hall.

Dix, C. H., 1955, Seismic velocities from surface measurements: Geophysics, **20**, 68–86.

Docherty, P., 1991, A brief comparison of some Kirchhoff integral formulas for migration and inversion: Geophysics, **56**, 1164–1169.

Durran, D. R., 1999, Numerical Methods for Wave Equations in Geophysical Fluid Dynamics: Springer.

Etter, D. E., 1996, Introduction to Matlab for Engineers and Scientists: Prentice-Hall.

Futterman, W. I., 1962, Dispersive body waves: J. Geophys. Res., **73**, 3917–3935.

Gallant, E. V., Bertram, M. B., and Stewart, R. R., 2002, Elastic-wave seismic acquisition systems: CREWES Research Report, **14**.

Gazdag, J., 1978, Wave-equation migration with the phase-shift method: Geophysics, **43**, 1342–1351.

Gray, S. H., 1986, Efficient traveltime calculations for Kirchhoff migration: Geophysics, **51**, 1685–1688.

Gray, S. H., 1997, Trueamplitude seismic migration: A comparison of three approaches: Geophysics, **62**, 929–936.

Gustafson, K. E., 1987, Partial Differential Equations and Hilbert Space Methods: John Wiley and Sons.

Hagedoorn, J. G., 1954, A process of seismic reflection interpretation: Geophysical Prospecting, **2**, 85–127.

Hall, K. W., Gallant, E. V., McKellar, M., Ursenbach, C. P., and Stewart, R. R., 2001, Ultrasonic imaging of a heavy oil recovery model: CREWES Research Report, **13**.

Hanselman, D. C., and Littlefield, B., 1998, Mastering MATLAB 5: A Comprehensive Tutorial and Reference: Prentice Hall.

Higham, D. J., and Higham, N. J., 2000, MATLAB Guide: Society for Industrial and Applied Mathematics.

Hoffe, B. H., and Lines, L. R., 1998, Depth imaging of elastic wavefields - where P meets S: CREWES Research Report, **10**.

Jol, H. M., and Smith, D. G., 1991, Ground penetrating radar: a brief overview and case study: CREWES Research Report, **3**.

Karl, J. H., 1989, An Introduction to Digital Signal Processing: Academic Press, Inc.

Kjartansson, E., 1980, Constant Q-wave propagation and theory: J. Geophys. Res., **84**, 4737–4748.

Korner, T. W., 1988, Fourier Analysis: Cambridge University Press, Inc.

Lancaster, P., and Salkauskas, K., 1996, Transform Methods in Applied Mathematics: John Wiley & Sons, Inc.

Lawton, D., and Jol, H., 1994, Ground penetrating radar investigation of the Canada creosote site, Calgary: CSEG Recorder, **19**, 13–18.

Lawton, D. C., Stewart, R. R., and Bland, H. C., 2002, Multicomponent seismic survey at Jumpingpound, Alberta: CREWES Research Report, **14**.

Liner, C. T., and Gobeli, R., 1996, Bin size and linear $v(z)$: Expanded abstracts 66th Annual Mtg. Soc. Expl. Geophy.

Liner, C. T., and Gobeli, R., 1997, 3-d seismic survey design and linear $v(z)$: Expanded abstracts 67th Annual Mtg. Soc. Expl. Geophy.

Lines, L. R., Slawinski, R., and Bording, R. P., 1999, A recipe for stability of finite-difference wave-equation computations: Geophysics, **64**, 967–969.

Lines, L., 1991, Applications of tomography to borehole and reflection seismology: The Leading Edge, **7**, 11–17.

Lokshtanov, D., 2002, Removal of water layer multiples and peg-legs by wave-equation approach: CREWES Research Report, **14**.

LowenthalD., L. L. R. R., and Sherwood, J. W., 1976, The wave equation applied to migration: Geophysical Prospecting, **24**, 380–399.

Lynn, H. B., and Deregowski, S., 1981, Dip limitations on migrated sections as a function of line length and recording time: Geophysics, **46**, 1392–1397.

Mazur, M. J., and Stewart, R. R., 1997, The seismic expression and hydrocarbon potential of meteorite impact craters: Current research: CREWES Research Report, **9**.

Mitchell, A. R., and Griffiths, D. F., 1980, The Finite Difference Method in Partial Differential Equations: John Wiley and Sons.

Morse, P. M., and Feshbach, H., 1953, Methods of Theoretical Physics: McGraw-Hill.

Press, W., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., 1992, Numerical Recipes in C: The Art of Scientific Computing: Cambridge University Press.

Redfern, D., and Campbell, C., 1998, The Matlab 5 Handbook: Springer-Verlag.

Robinson, E. A., 1983, Migration of Geophysical Data: International Human Resources Development Corp. (IHRDC).

Scales, J. A., 1995, Theory of Seismic Imaging: Springer.

Schneider, W. S., 1978, Integral formulation for migration in two or three dimensions: Geophysics, **43**, 49–76.

Schuster, G. T., 1997, Green's function for migration: Expanded Abstracts 67th Annual Mtg. Soc. Expl. Geophy., pages 1754–1757.

Sheriff, R. E., and Geldart, L. P., 1995, Exploration Seismology, Second Edition: Cambridge University Press.

Simin, V., Harrison, M. P., and Lorentz, G. A., 1996, Processing the Blackfoot 3C-3D seismic survey: CREWES Research Report, **8**.

Slotnick, M. M., 1959, Lessons in Seismic Computing: Society of Exploration Geophysicists.

Stein, E. M., 1993, Harmonic Analysis: real-variable methods, orthogonality, and oscillatory integrals: Princeton University Press, Inc.

Stolt, R. H., 1978, Migration by Fourier transform: Geophysics, **43**, 23–48.

Taner, M. T., and Koehler, F., 1969, Velocity spectra-digital computer derivation and applications of velocity functions: Geophysics, **34**, 859–881.

Taner, M. T., Koehler, F., and Sheriff, R. E., 1979, Complex seismic trace analysis: Geophysics, **44**, 1041–63.

Taylor, M. E., 1996, Partial Differential Equations (3 volumes): Springer.

Vermeer, G. J. O., 1990, Seismic Wavefield Sampling: Society of Exploration Geophysicists.

Wiggins, J. W., 1984, Kirchhoff integral extrapolation an migration of nonplanar data: Geophysics, **49**, 1239–1248.

Yan, L., and Lines, L., 2001, Seismic imaging and velocity analysis for an Alberta Foothills seismic survey: Geophysics, **66**, 721–732.

Zauderer, E., 1989, Partial Differential Equations of Applied Mathematics: John Wiley and Sons.

Index