1 Introduction to MATLAB and Seismic Data

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 toward 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 experience 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, multimillion-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 through the computer exercises, requires access to a significant machine (at least a late-model PC or Macintosh) with MATLAB installed and having significant memory and disk drive capacity.

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 more 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

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

datasets. As with many MATLAB applications, it helps greatly if the reader has had some experience with linear algebra. Hopefully, the concepts of matrices, row vectors, column vectors, 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 1980s, 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, these authors included. MATLAB evolved from the Linpack package, which 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 in 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, which does not require so many explicit loops. For someone well versed in Fortran, it can be difficult to unlearn 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 toward 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.

1.2 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 *NMES Toolbox* refers to the entire collection of software provided in this book.

MATLAB code will be presented in small numbered packages entitled "code snippets." An example is the code required to convert an amplitude spectrum from linear to decibel scale (Code Snippet 1.2.1).

The actual MATLAB code is displayed in an upright typewriter font, while introductory remarks are *emphasized like this*. The code snippets do not employ typographic variations to indicate which functions are contained in the *NMES Toolbox* as is done in the text proper.

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. The final line prints the figure into an "eps" file, a detail that will not usually be included in subsequent code snippets.

```
1   [wavem,t]=wavemin(.001,20,1);
2   [Wavem,f]=fftrl(wavem,t);
3   Amp=abs(Wavem);
4   dbAmp=20*log10(Amp/max(Amp));
5   figure
6   subplot(3,1,1);plot(t,wavem);xlabel('time (sec)');xlim([0 .2])
7   subplot(3,1,2);plot(f,abs(Amp));xlabel('Hz');ylabel('linear scale');
8   ylim([0 1.01])
9   subplot(3,1,3);plot(f,dbAmp);xlabel('Hz');ylabel('db down')
10  prepfiga
11  bigfont(gcf,2,1)
12
13  print -deps .\intrographics\intro1a.eps
```

End Code

introcode / intro1 .m

It has proven impractical to discuss all of the input parameters for all of the programs shown in the 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.[2] For example, in Code Snippet 1.2.1, *wavemin* creates a minimum-phase wavelet sampled at 0.002 s, with a dominant frequency of 20 Hz and a length of 0.2 s. 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) on line 4.

A decibel scale is a logarithmic display of amplitude, which is extremely useful when the amplitude range extends over many orders of magnitude. Given an amplitude function like $A(f)$, where in this context we are speaking of the amplitude of a spectrum as a function of frequency, the decibel representation comes from the formula

$$A_{\mathrm{dB}}(f) = 20\log_{10}\left(\frac{A(f)}{A_{\mathrm{ref}}}\right), \tag{1.1}$$

where we assume $A(f) > 0$ everywhere (if it is not obvious why, then you might want to review logarithms), and A_{ref} is a reference amplitude. Often the choice $A_{\mathrm{ref}} = \max(A(f))$ is made, which means that $A_{\mathrm{dB}}(f)$ will be all negative numbers except at the maximum, where it will be zero. Suppose $A(f) = A_{\mathrm{ref}}/2$; then $A_{\mathrm{dB}} = 20\log_{10}(0.5) \approx -6$ dB. Using the properties of logarithms, if $A(f) = A_{\mathrm{ref}}/4$, then $A_{\mathrm{dB}} = 20\log_{10}(\frac{1}{2}\frac{1}{2}) = 20[\log_{10}\frac{1}{2} + \log_{10}\frac{1}{2}] \approx -12$ dB. So, every -6 dB represents a decrease in amplitude by a factor of $\frac{1}{2}$. Similarly, if the reference amplitude is not the maximum, then every increase in amplitude

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

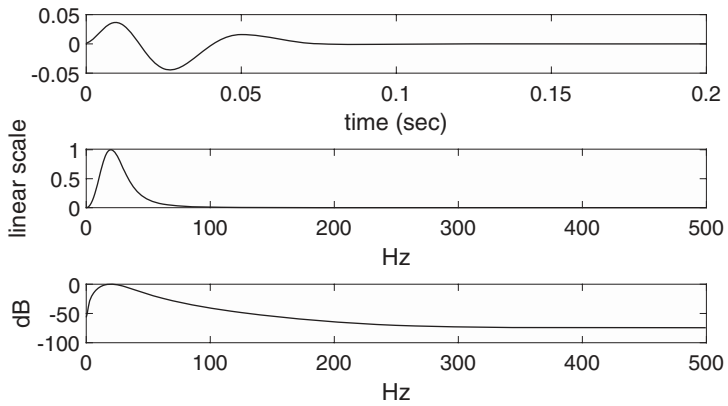

Figure 1.1 A minimum-phase wavelet (top), its amplitude spectrum plotted on a linear scale (middle), and its amplitude spectrum plotted on a decibel scale (bottom).

by a factor of 2 is a +6 dB increase. Sometimes we prefer to think in factors of 10, and it is similarly easy to show that -20 dB represents a decrease by a factor of $\frac{1}{10}$, while +20 dB is an increase by 10.

This example illustrates several additional conventions. Digital seismic traces[3] are discrete time series, but the common 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 vector. 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 top graph 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 hertz. (One hertz (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 formulas ($\omega = 2\pi f$).

Seismic traces will always be represented by column vectors, whether in the time or the Fourier domain. This allows an easy transition to 2D 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

[3] A *seismic trace* is usually the recording of a single-component geophone or hydrophone. Less commonly, for a multicomponent geophone, it is the recording of one component.

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 boxcar window. (A boxcar window is a time series that is all zeros except for a selected range, where it is 1.0.)

```
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=0*(1:length(s));%initialize window to all zeros
6   win(n2-50:n2+50)=1;%100 samples in the center of win are 1.0
7   swin=s.*win; % apply the window to the trace
```

End Code

introcode / intro2 .m

each sample in the trace by the corresponding sample in the window. The temptation is to write a code something like Code Snippet 1.2.2.

MATLAB's response to this code is the error message

```
Error using  .*
Matrix dimensions must agree.
```

The error occurs because the code `1:length(s)` used on line 5 generates a 1×501 row vector like $1, 2, 3, \ldots, n$, where `n=length(s)`, while `s` is a 501×1 column vector. The .* operator requires both operands to 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 38 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 not directly relevant or 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.

1.3 Seismic Wavelets

In the previous section, mention was made of a *minimum-phase wavelet*, of which there is a great variety, and there are also many other wavelet types. In seismology, a wavelet typically represents the waveform emitted by a seismic source and as possibly modified by data processing. All wavelet types have a specific mathematical form, which may have

variable parameters such as a dominant frequency, a wavelet length (in time), and perhaps a phase option. The mathematical formulas that these wavelets are based on will be presented later, in Section 4.7.1. For now, we will simply illustrate how to generate some basic wavelets and discuss their most obvious properties.

A minimum-phase wavelet (Figure 1.1) refers to a specific type of wavelet that is thought to be a good model for the waveform from an impulsive seismic source such as dynamite. For now, the reasons for the name "minimum phase" will not be investigated; rather, it is sufficient to realize that such wavelets are causal,[4] have a shaped amplitude spectrum with a dominant frequency, have an amplitude spectrum that must never vanish at any frequency,[5] and have a phase spectrum that is linked to the shape of the amplitude spectrum. (If these terms such as "spectrum," "amplitude," and "phase" are unfamiliar, rest assured that they will be explained fully in Chapter 2.) Minimum-phase wavelets are typically used to model raw or unprocessed data. While they arise naturally, they are not usually preferred for interpretation, because their peak energy is always delayed from $t = 0$. It is a major task of data processing to estimate the wavelet in the raw data (it is never known a priori) and design an operator to shape it to zero phase. Thus various zero-phase wavelets are typically used to represent processed seismic data.

Figures 1.2a and 1.2b show four typical wavelets in the time and frequency domains. The figures were created by Code Snippet 1.3.1. The minimum-phase wavelet, created by `wavemin`, shows the basic causal nature and the delay of the maximum energy from $t = 0$. The Ricker wavelet, created by `ricker`, is a popular wavelet for use in seismic interpretation. A frequent task is to create a synthetic seismogram from well information to compare with a final seismic image. The synthetic seismogram, s, is usually constructed from a convolutional model $s(t) = w(t) \bullet r(t)$, where r is the reflectivity function (derived from logging information), w is usually a zero-phase wavelet, very often a Ricker wavelet, and $\bullet$ represents convolution.[6] In Figure 1.2b, the amplitude spectra of these wavelets are shown, and the minimum-phase and Ricker wavelets are similar, with smoothly sloping spectra away from a dominant frequency. While the Ricker spectrum is only adjustable by shifting the dominant frequency, the decay rate of the minimum-phase wavelet's spectrum can be controlled with the parameter `m` on line 8. The other two wavelets, Ormsby (from `ormsby`) and Klauder (from `klauder`), have no single dominant frequency but instead have a broad, flat passband. The details of these wavelets and spectra can be controlled from the input parameters in the various programs. In Code Snippet 1.3.1, these parameters have been deliberately chosen to make the wavelets similar.

Comparing the different zero-phase wavelets, it is apparent that the Ricker wavelet has the least sidelobe energy but also has a greater width as measured between the first zero crossings on either side of the maximum. The choice of which zero-phase wavelet to use in creating a seismogram will depend upon the nature of the data and the data processing. The Klauder wavelet is intended for *correlated* Vibroseis data. Vibroseis data is usually

[4] A causal wavelet is one that vanishes before a specific time, usually taken to be $t = 0$, and persists arbitrarily long afterwards.
[5] Except possibly at a few isolated points.
[6] Convolution is described intuitively in Section 2.3.1.

Code Snippet 1.3.1 This code creates Figures 1.2a and 1.2b. Four wavelets are created: minimum phase with *wavemin*, Ricker with *ricker*, Ormsby with *ormsby*, and Klauder with *klauder*. The minimum-phase and Ricker wavelets have the same dominant frequency, specified on line 2. The Ormsby and Klauder wavelets have broad, flat passbands from fmin to fmax. The wavelets all have the same temporal length of tlen, which is set here to 2.0 s. This is much longer than usual and is done simply to force fine sampling in the frequency domain and make a better graph. The wavelets are created on lines 10–13 and their spectra are calculated on lines 15–18 using *fftrl*. Note the use of *linesgray* to plot, rather than *plot*. This makes figures with gray lines of different shades rather than color. The equivalent *plot* commands are shown commented out for comparison.

```
1   dt=.001;%time sample size
2   fdom=30;%dominant frequency for ricker and wavemin
3   fmin=10;%beginning of passband for ormsby and klauder
4   fmax=70;%end of passband for ormsby and klauder
5   tlen=2;%temporal length of each wavelet
6   slen=8;%sweep length (klauder)
7   staper=.5;%sweep taper (klauder)
8   m=3.5;%spectral decay control in wavemin
9
10  [wm,twm]=wavemin(dt,fdom,tlen,m);
11  [wr,twr]=ricker(dt,fdom,tlen);
12  [wo,two]=ormsby(fmin-5,fmin,fmax,fmax+10,tlen,dt);
13  [wk,twk]=klauder(fmin,fmax,dt,slen,tlen,staper);
14
15  [Wm,fwm]=fftrl(wm,twm);
16  [Wr,fwr]=fftrl(wr,twr);
17  [Wo,fwo]=fftrl(wo,two);
18  [Wk,fwk]=fftrl(wk,twk);
19
20  figure
21  inc=.1;
22  %plot(twm,wm,twr,wr+inc,two,wo+2*inc,twk,wk+3*inc);
23  linesgray({twm,wm,'-',.4,0},{twr,wr+inc,'-',.9,.3},...
24      {two,wo+2*inc,'-',.9,.5},{twk,wk+3*inc,'-',1.1,.7});
25  xlabel('time (sec)')
26  xlim([-.25 .25]);ylim([-.1 .45])
27  legend('Minimum phase','Ricker','Ormsby','Klauder')
28  grid
29  prepfig;bigfont(gca,1.8,1)
30
31  figure
32  %plot(fwm,abs(Wm),fwr,abs(Wr),fwo,abs(Wo),fwk,abs(Wk))
33  linesgray({fwm,abs(Wm),'-',.4,0},{fwr,abs(Wr),'-',.9,.3},...
34      {fwo,abs(Wo),'-',.9,.5},{fwk,abs(Wk),'-',1.1,.7});
35  xlabel('frequency (Hz)')
36  xtick([0 10 30 50 70 100 150])
37  legend('Minimum phase','Ricker','Ormsby','Klauder')
38  xlim([0 150]);ylim([0 1.1])
39  grid
40  prepfig;bigfont(gca,1.8,1)
41
```

End Code

introcode / waveletsfig .m

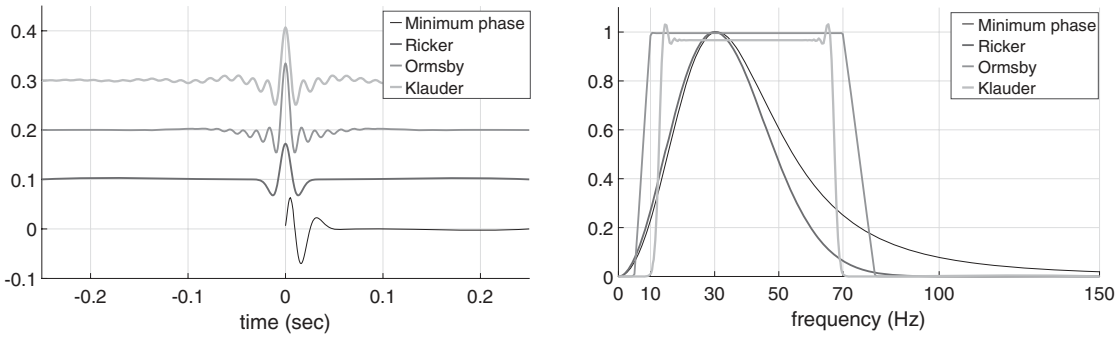

Figure 1.2a (left) Four common seismic wavelets. The minimum-phase wavelet is a causal wavelet, which is a model for an impulsive source like dynamite. The Ricker wavelet is zero phase with minimal sidelobes. The Ormsby wavelet is also zero phase but with much larger sidelobes. The zero-phase Klauder wavelet is a common model for Vibroseis data. The wavelets are plotted with vertical shifts to avoid plotting on top of each other.

Figure 1.2b (right) The amplitude spectra of the wavelets of Figure 1.2a. The spectra of the minimum-phase and Ricker wavelets decay smoothly away from a dominant frequency. The Ormsby and Klauder wavelets have a flat, broad passband and then decay sharply at the limits of the passband. The ears at the edges of the Klauder passband are related to the length of the sweep taper.

acquired with a source signature known as a sweep. A sweep is a temporally long[7] signal in which the vibrator runs through a prescribed range of frequencies called the *swept band*. When equal power is given to each frequency, the sweep is known as a linear sweep, and this is the usual case. Such data can be modeled as $s_v(t) = \sigma(t) \bullet I(t)$, where $\sigma(t)$ is the sweep and $I(t)$ is the Earth's impulse response that would be recorded from a perfectly impulsive source. After acquisition, the raw data is crosscorrelated with the sweep. We can represent the correlated data as $s_c = \sigma(t) \otimes s_v(t) = w_k(t) \bullet I(t)$, where $\otimes$ means crosscorrelation and w_k is the Klauder wavelet defined by $w_k(t) = \sigma(t) \otimes \sigma(t)$. Thus the Klauder wavelet is the crosscorrelation of the sweep with itself, or the *autocorrelation* of the sweep. There is a theoretical formula for the Klauder wavelet, but it is more realistic to actually construct the sweep and numerically autocorrelate it, and this is what `klauder` does. In Figure 1.2b, the Klauder spectrum is noticeably more narrow than the Ormsby one. This is because, while the sweep begins and ends at 10 and 70 Hz, respectively, its amplitude must rise gradually from zero to full strength because it is used to drive a large machine. The length of time over which this happens is called the sweep taper and is applied at both ends of the sweep. The taper causes reduced power at the beginning and end of the sweep and is also responsible for the "ears" seen at about 15 and 65 Hz. Reducing the length of the taper will broaden the spectrum but increase the height of the ears.

The Ormsby wavelet is often used to represent seismic data that has been deliberately processed to have a flat passband over some prescribed range. This produces a very narrow central wavelet peak but with quite a lot of sidelobe energy. The specification of an Ormsby

[7] Sweep lengths ranging from 4 s to as long as 20 s are common in practice.

wavelet is usually done by giving the four *Ormsby parameters*, which are called f_1, f_2, f_3, f_4, with the following meanings:

- f_1: high end of the low-frequency stop band. There is no signal for $f < f_1$.
- f_2: low end of the passband. The amplitude ramps up linearly for $f_1 \le f \le f_2$.
- f_3: high end of the passband. The amplitude is constant for $f_2 \le f \le f_3$.
- f_4: low end of the high-frequency stop band. There is no signal for $f > f_4$.

The theoretical Ormsby wavelet is infinitely long, and any practical version must be truncated. This truncation causes the stop bands to have some nonzero energy.

Each of these wavelets is normalized such that a sine wave of the dominant frequency is unchanged in amplitude when convolved with the wavelet.

1.4 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 a fixed instrument gain and represent amplitudes as 24-bit integer computer words.[8] The first bit is used to record the 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 Eq. (1.1), this corresponds to about 132 dB, an enormous spread of possible values. This 132 dB range is never usually 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 affect only 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 the smallest number in the dynamic range.

1.4.1 Single-Trace Plotting and Dynamic Range

Figure 1.3 was produced with Code Snippet 1.4.1 and shows two real seismic traces recorded in 1997 by CREWES.[9] This type of plot is called a *wiggle trace display*.

[8] 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.

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