Chapter 1

Waves and Fourier sums

An important concept in wave imaging is the extrapolation of a wavefield from one z to another. Fourier transforms are an essential basic tool. There are many books and chapters of books on the *theory* of Fourier transformation. The first half of this chapter is an introduction to *practice* with Fourier sums. It assumes you already know something of the theory and takes you through the theory rather quickly emphasizing practice by examining examples, and by performing two-dimensional Fourier transformation of data and interpreting the result. For a somewhat more theoretical background, I suggest my previous book PVI at http://sepwww.stanford.edu/sep/prof/.

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

1.1 FOURIER TRANSFORM

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

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

Look what happens to the coefficients when we multiply polynomials.

$$X(Z)\,B(Z) \quad = \quad Y(Z) \tag{1.1}$$

$$(x_0 + x_1 Z + x_2 Z^2 + \cdots)(b_0 + b_1 Z + b_2 Z^2) \quad = \quad y_0 + y_1 Z + y_2 Z^2 + \cdots \tag{1.2}$$

Identifying coefficients of successive powers of Z, we get

$$\begin{aligned} y_0 &= x_0 b_0 \\ y_1 &= x_1 b_0 + x_0 b_1 \\ y_2 &= x_2 b_0 + x_1 b_1 + x_0 b_2 \end{aligned} \tag{1.3}$$

$$\begin{aligned} y_3 &= x_3b_0 + x_2b_1 + x_1b_2 \\ y_4 &= x_4b_0 + x_3b_1 + x_2b_2 \\ &= \cdots\cdots\cdots\cdots \end{aligned}$$

In matrix form this looks like

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} x_0 & 0 & 0 \\ x_1 & x_0 & 0 \\ x_2 & x_1 & x_0 \\ x_3 & x_2 & x_1 \\ x_4 & x_3 & x_2 \\ 0 & x_4 & x_3 \\ 0 & 0 & x_4 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \tag{1.4}$$

The following equation, called the "convolution equation," carries the spirit of the group shown in (1.3)

$$y_k = \sum_{i=0} x_{k-i} b_i \tag{1.5}$$

The second way to visualize polynomial multiplication is simpler. Above we did not think of Z as a numerical value. Instead we thought of it as "a unit delay operator". Now we think of the product $X(Z)B(Z) = Y(Z)$ numerically. For all possible numerical values of Z, each value Y is determined from the product of the two numbers X and B. Instead of considering all possible numerical values we limit ourselves to all values of unit magnitude $Z = e^{i\omega}$ for all real values of ω. This is Fourier analysis, a topic we consider next.

1.1.1 FT as an invertible matrix

A **Fourier sum** may be written

$$B(\omega) = \sum_t b_t \, e^{i\omega t} = \sum_t b_t \, Z^t \tag{1.6}$$

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

$$B(\omega) = b_0 + b_1 Z + b_2 Z^2 + b_3 Z^3 \tag{1.7}$$

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

$$\begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{1.8}$$

Observe that the top row of the matrix evaluates the polynomial at $Z = 1$, a point where also $\omega = 0$. The second row evaluates $B_1 = B(Z = W = e^{i\omega_0})$, where ω_0 is some base frequency. The third row evaluates the Fourier transform for $2\omega_0$, and the bottom row for $3\omega_0$. The matrix could have more than four rows for more frequencies and more columns for more time points. I have made the matrix square in order to show you next how we can find the inverse matrix. The size of the matrix in (1.8) is $N = 4$. If we choose the base frequency ω_0 and hence W correctly, the inverse matrix will be

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = 1/N \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/W & 1/W^2 & 1/W^3 \\ 1 & 1/W^2 & 1/W^4 & 1/W^6 \\ 1 & 1/W^3 & 1/W^6 & 1/W^9 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} \tag{1.9}$$

Multiplying the matrix of (1.9) with that of (1.8), we first see that the diagonals are +1 as desired. To have the off diagonals vanish, we need various sums, such as $1 + W + W^2 + W^3$ and $1 + W^2 + W^4 + W^6$, to vanish. Every element (W^6, for example, or $1/W^9$) is a unit vector in the complex plane. In order for the sums of the unit vectors to vanish, we must ensure that the vectors pull symmetrically away from the origin. A uniform distribution of directions meets this requirement. In other words, W should be the N-th root of unity, i.e.,

$$W = \sqrt[N]{1} = e^{2\pi i/N} \tag{1.10}$$

The lowest frequency is zero, corresponding to the top row of (1.8). The next-to-the-lowest frequency we find by setting W in (1.10) to $Z = e^{i\omega_0}$. So $\omega_0 = 2\pi/N$; and for (1.9) to be inverse to (1.8), the frequencies required are

$$\omega_k = \frac{(0, 1, 2, \ldots, N-1)2\pi}{N} \tag{1.11}$$

1.1.2 The Nyquist frequency

The highest frequency in equation (1.11), $\omega = 2\pi(N-1)/N$, is almost 2π. This frequency is twice as high as the Nyquist frequency $\omega = \pi$. The **Nyquist frequency** is normally thought of as the "highest possible" frequency, because $e^{i\pi t}$, for integer t, plots as $(\cdots, 1, -1, 1, -1, 1, -1, \cdots)$. The double Nyquist frequency function, $e^{i2\pi t}$, for integer t, plots as $(\cdots, 1, 1, 1, 1, 1, 1, \cdots)$. So this frequency above the highest frequency is really zero frequency! We need to recall that $B(\omega) = B(\omega - 2\pi)$. Thus, all the frequencies near the upper end of the range equation (1.11) are really small negative frequencies. Negative frequencies on the interval $(-\pi, 0)$ were moved to interval $(\pi, 2\pi)$ by the matrix form of Fourier summation.

A picture of the Fourier transform matrix is shown in Figure 1.1. Notice the Nyquist frequency is the center row and center column of each matrix.

1.1.3 Laying out a mesh

In theoretical work and in programs, the unit delay operator definition $Z = e^{i\omega\Delta t}$ is often simplified to $\Delta t = 1$, leaving us with $Z = e^{i\omega}$. How do we know whether ω is given in radians

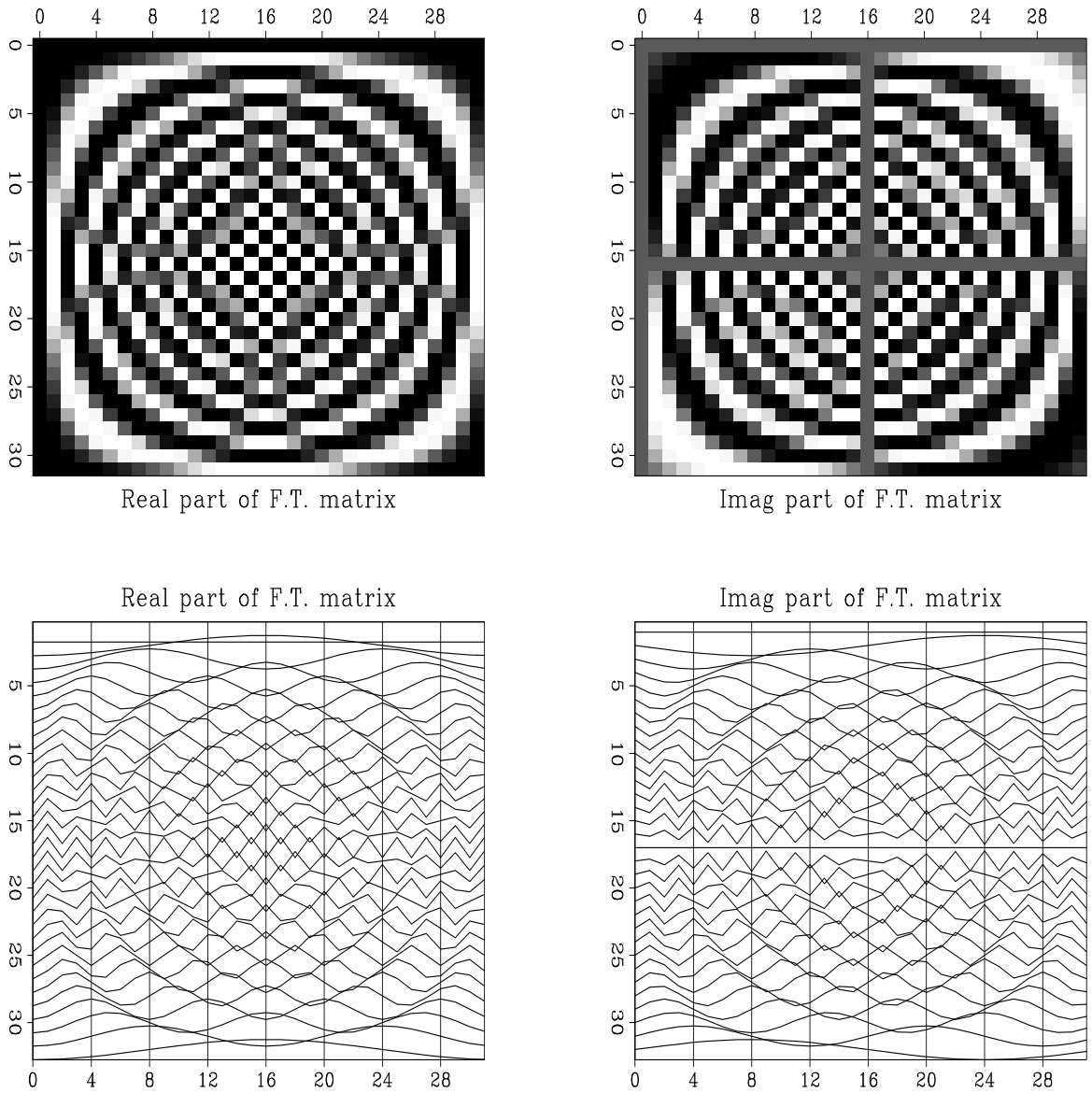

Figure 1.1: Two different graphical means of showing the real and imaginary parts of the Fourier transform matrix of size 32×32. ft1-matrix [ER]

per second or radians per sample? We may not invoke a cosine or an exponential unless the argument has no physical dimensions. So where we see ω without Δt, we know it is in units of radians per sample.

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

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

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

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

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

$$\Delta t \; f_{\text{range}} = 1 \tag{1.12}$$

$$\Delta f \; t_{\text{range}} = 1 \tag{1.13}$$

$$\Delta f \; \Delta t = \frac{1}{N} \tag{1.14}$$

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

1.2 INVERTIBLE SLOW FT PROGRAM

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

There is no universally correct choice of **scale factor** in Fourier transform: choice of scale is a matter of convenience. Equations (1.8) and (1.9) mimic the Z-transform, so their scaling factors are convenient for the convolution theorem—that a product in the frequency domain is a convolution in the time domain. Obviously, the scaling factors of equations (1.8) and (1.9) will need to be interchanged for the complementary theorem that a convolution in the frequency domain is a product in the time domain. I like to use a scale factor that keeps the sums of squares the same in the time domain as in the frequency domain. Since I almost never need the scale factor, it simplifies life to omit it from the subroutine argument list. When a scaling program is desired, we can use a simple one like `scale()` on this page. Complex-valued data can be scaled with `scale()` merely by doubling the value of `n`.

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

1.2.1 The simple FT code

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

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

The total frequency band is 2π radians per sample unit or $1/\Delta t$ Hz. Dividing the total interval by the number of points `nf` gives Δf. We could choose the frequencies to run from 0 to 2π radians/sample. That would work well for many applications, but it would be a nuisance for

applications such as differentiation in the frequency domain, which require multiplication by $-i\omega$ including the **negative frequencies** as well as the positive. So it seems more natural to begin at the most negative frequency and step forward to the most positive frequency.

1.3 CORRELATION AND SPECTRA

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

1.3.1 Spectra in terms of Z-transforms

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

$$S(\omega) \quad = \quad |B(\omega)|^2 \quad = \quad \overline{B(\omega)}B(\omega) \tag{1.15}$$

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

$$S(\omega) \quad = \quad (\bar{b}_0+\bar{b}_1e^{-i\omega}+\bar{b}_2e^{-i2\omega})(b_0+b_1e^{i\omega}+b_2e^{i2\omega}) \tag{1.16}$$

$$S(Z) \quad = \quad \left(\bar{b}_0+\frac{\bar{b}_1}{Z}+\frac{\bar{b}_2}{Z^2}\right)(b_0+b_1Z+b_2Z^2) \tag{1.17}$$

$$S(Z) \quad = \quad \overline{B}\left(\frac{1}{Z}\right)B(Z) \tag{1.18}$$

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

$$S(Z) \quad = \quad \frac{\bar{b}_2b_0}{Z^2}+\frac{(\bar{b}_1b_0+\bar{b}_2b_1)}{Z}+(\bar{b}_0b_0+\bar{b}_1b_1+\bar{b}_2b_2)+(\bar{b}_0b_1+\bar{b}_1b_2)Z+\bar{b}_0b_2Z^2$$

$$S(Z) \quad = \quad \frac{s_{-2}}{Z^2}+\frac{s_{-1}}{Z}+s_0+s_1Z+s_2Z^2 \tag{1.19}$$

The coefficient s_k of Z^k is given by

$$s_k \quad = \quad \sum_i \bar{b}_ib_{i+k} \tag{1.20}$$

Equation (1.20) is the **autocorrelation** formula. The autocorrelation value s_k at lag 10 is s_{10}. It is a measure of the similarity of b_i with itself shifted 10 units in time. In the most frequently occurring case, b_i is real; then, by inspection of (1.20), we see that the autocorrelation coefficients are real, and $s_k = s_{-k}$.

Specializing to a real time series gives

$$S(Z) \quad = \quad s_0+s_1\left(Z+\frac{1}{Z}\right)+s_2\left(Z^2+\frac{1}{Z^2}\right) \tag{1.21}$$

$$
\begin{aligned}
S(Z(\omega)) &= s_0 + s_1(e^{i\omega} + e^{-i\omega}) + s_2(e^{i2\omega} + e^{-i2\omega}) && (1.22)\\
S(\omega) &= s_0 + 2s_1\cos\omega + 2s_2\cos 2\omega && (1.23)\\
S(\omega) &= \sum_k s_k \cos k\omega && (1.24)\\
S(\omega) &= \text{cosine transform of } \; s_k && (1.25)
\end{aligned}
$$

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

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

1.3.2 Two ways to compute a spectrum

There are two computationally distinct methods by which we can compute a spectrum: (1) compute all the s_k coefficients from (1.20) and then form the cosine sum (1.24) for each ω; and alternately, (2) evaluate $B(Z)$ for some value of Z on the unit circle, and multiply the resulting number by its complex conjugate. Repeat for many values of Z on the unit circle. When there are more than about twenty lags, method (2) is cheaper, because the fast Fourier transform (coming up soon) can be used.

1.3.3 Common signals

Figure 1.2 shows some common signals and their **autocorrelation**s. Figure 1.3 shows the cosine transforms of the autocorrelations. Cosine transform takes us from time to frequency and it also takes us from frequency to time. Thus, transform pairs in Figure 1.3 are sometimes more comprehensible if you interchange time and frequency. The various signals are given names in the figures, and a description of each follows:

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

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

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

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

twin Two pulses.

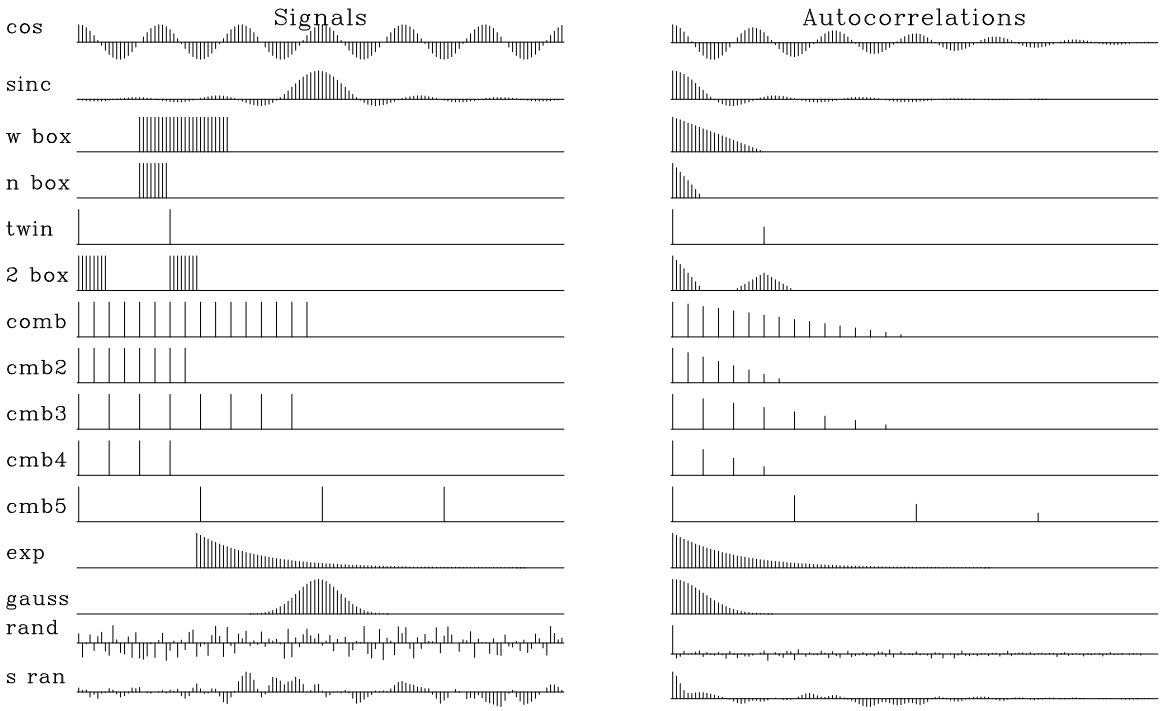

Figure 1.2: Common signals and one side of their autocorrelations. ft1-autocor [ER]

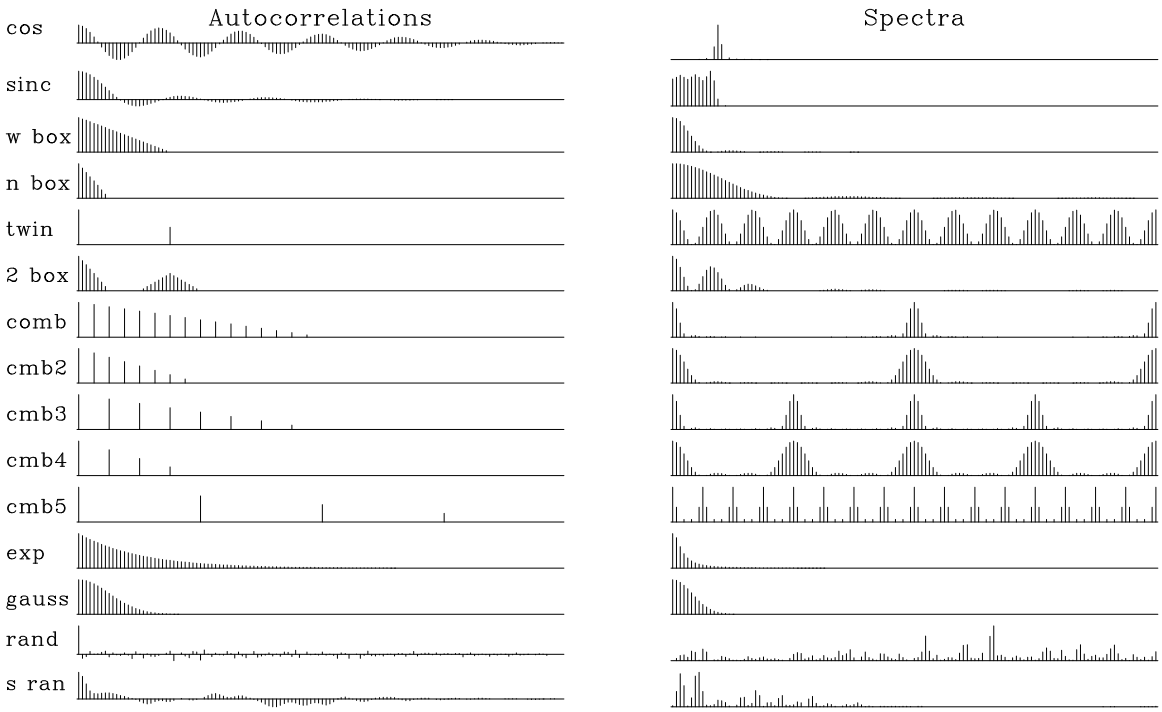

Figure 1.3: Autocorrelations and their cosine transforms, i.e., the (energy) spectra of the common signals. ft1-spectra [ER]

2 boxes Two separated narrow boxes have the spectrum of one of them, but this spectrum is modulated (multiplied) by a sinusoidal function of frequency, where the modulation frequency measures the time separation of the narrow boxes. (An oscillation seen in the frequency domain is sometimes called a "**quefrency**.")

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

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

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

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

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

1.4 SETTING UP THE FAST FOURIER TRANSFORM

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

Flexibility may be lost because the basic fast program works with complex-valued signals, so we ordinarily convert our real signals to complex ones (by adding a zero imaginary part). More flexibility is lost because typical fast FT programs require the data length to be an integral power of 2. Thus geophysical datasets often have zeros appended (a process called "**zero pad**ding") until the data length is a power of 2. From time to time I notice clumsy computer code written to deduce a number that is a power of 2 and is larger than the length of a dataset. An answer is found by rounding up the logarithm to base 2. The more obvious and the quicker way to get the desired value, however, is with the simple Fortran function `pad2()`.

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

How fast is the fast Fourier transform method? The answer depends on the size of the data. The matrix times vector operation in (1.8) requires N^2 multiplications and additions. That determines the speed of the slow transform. For the fast method the number of adds and multiplies is proportional to $N \log_2 N$. Since $2^{10} = 1024$, the speed ratio is typically 1024/10 or about 100. In reality, the fast method is not quite that fast, depending on certain details of overhead and implementation.

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

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

The following two lines serve to Fourier transform a vector of 1024 complex-valued points, and then to **inverse Fourier transform** them back to the original data:

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

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

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

1.4.1 Shifted spectrum

Subroutine `simpleft()` on page 6 sets things up in a convenient manner: The frequency range runs from minus Nyquist up to (but not including) plus Nyquist. Thus there is no problem with the many (but not all) user programs that have trouble with aliased frequencies. Subroutine `ftu()` on the page before, however has a frequency range from zero to double the Nyquist. Let us therefore define a friendlier "front end" to `ftu()` which looks more like `simpleft()`.

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

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

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

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

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

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

1.5 SETTING UP 2-D FT

The program `fth()` is set up so that the vectors transformed can be either rows or columns of a two-dimensional array. In any computer language there is a way to extract a vector (column or row) from a matrix. In some languages the vector can be processed directly without extraction. To see how this works in **Fortran**, recall a matrix allocated as `(n1,n2)` can be subscripted as a matrix `(i1,i2)` or as a long vector `(i1 + n1*(i2-1),1)`, and `call sub(x(i1,i2))` passes the subroutine a pointer to the `(i1,i2)` element. To transform an entire axis, the subroutines `ft1axis()` and `ft2axis()` are given. For a two-dimensional FT, we simply call both `ft1axis()` and `ft2axis()` in either order.

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


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

1.5.1 Basics of two-dimensional Fourier transform

Let us review some basic facts about **two-dimensional Fourier transform**. A two-dimensional function is represented in a computer as numerical values in a matrix, whereas a one-dimensional

Fourier transform in a computer is an operation on a vector. A 2-D Fourier transform can be computed by a sequence of 1-D Fourier transforms. We can first transform each column vector of the matrix and then each row vector of the matrix. Alternately, we can first do the rows and later do the columns. This is diagrammed as follows:

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

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

An example of **two-dimensional Fourier transform**s on typical deep-ocean data is shown in Figure 1.4. In the deep ocean, sediments are fine-grained and deposit slowly in flat, regular, horizontal beds. The lack of permeable rocks such as sandstone severely reduces the potential for petroleum production from the deep ocean. The fine-grained shales overlay irregular, igneous, **basement rock**s. In the plot of $P(t,k_x)$, the lateral continuity of the sediments is shown by the strong spectrum at low k_x. The igneous rocks show a k_x spectrum extending to such large k_x that the deep data may be somewhat **spatially aliased** (sampled too coarsely). The plot of $P(\omega,x)$ shows that the data contains no low-frequency energy. The dip of the sea floor shows up in (ω,k_x)-space as the energy crossing the origin at an angle.

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

$$p(t,x) \;=\; \int\int e^{-i\omega t + ik_x x}\; P(\omega,k_x)\; d\omega\; dk_x \tag{1.26}$$

The double integration can be nested to show that the temporal transforms are done first (inside):

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

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

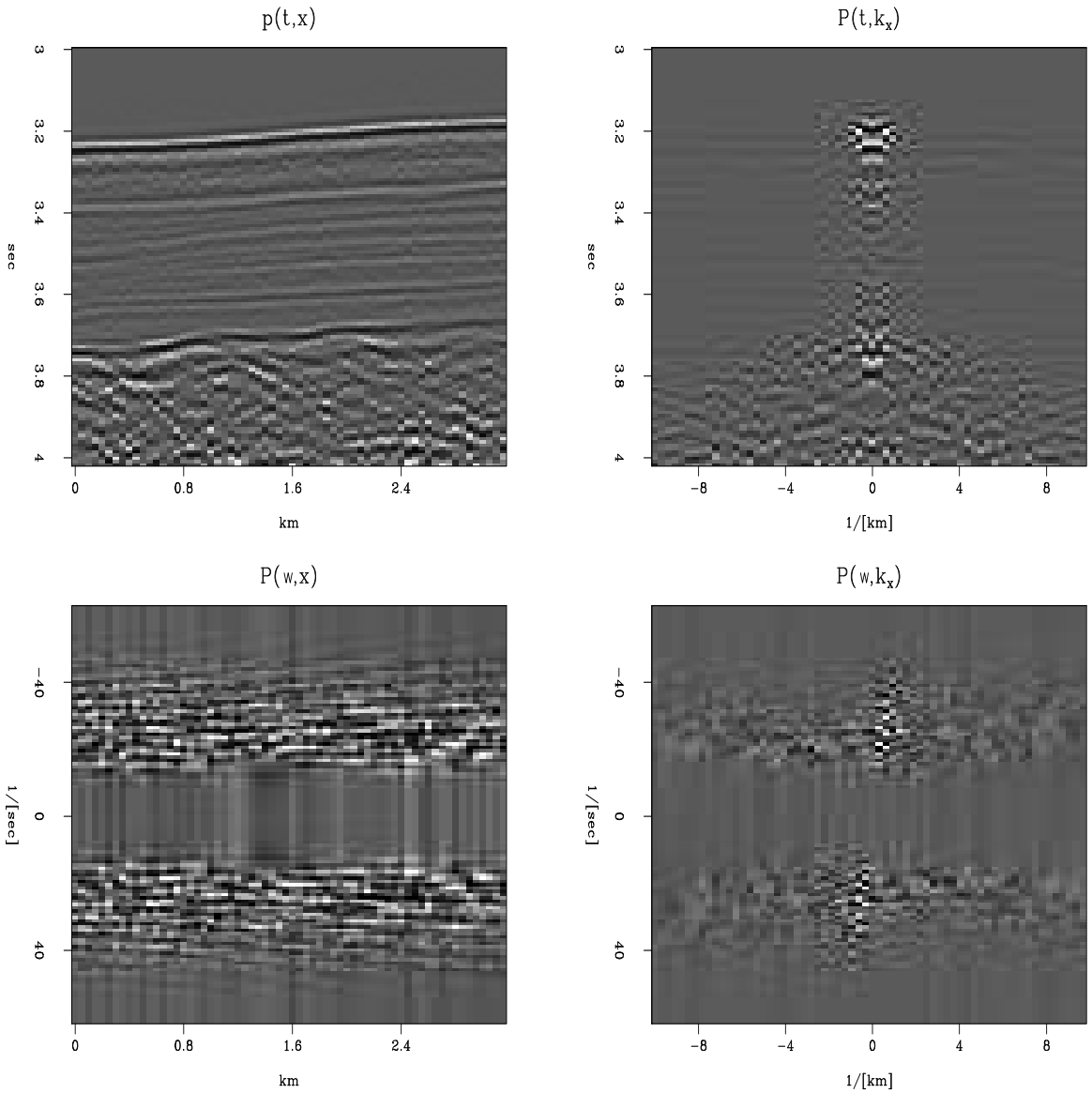

Figure 1.4: A deep-marine dataset $p(t,x)$ from Alaska (U.S. Geological Survey) and the *real* part of various Fourier transforms of it. Because of the long traveltime through the water, the time axis does not begin at $t = 0$. ft1-plane4 [ER]

1.5.2 Signs in Fourier transforms

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

$$p(t,x,z) \quad = \quad \int\int\int e^{-i\omega t + ik_x x + ik_z z} \; P(\omega,k_x,k_z) \; d\omega \, dk_x \, dk_z \tag{1.27}$$

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

Let us see the reasons why electrical engineers have made the opposite choice, and why we go with the physicists. Essentially, engineers transform only the time axis, whereas physicists transform both time and space axes. Both are simplifying their lives by their choice of sign convention, but physicists complicate their time axis in order to simplify their many space axes. The engineering choice minimizes the number of minus signs associated with the time axis, because for engineers, d/dt is associated with $i\omega$ instead of, as is the case for us and for physicists, with $-i\omega$. We confirm this with equation (1.27). Physicists and geophysicists deal with many more independent variables than time. Besides the obvious three space axes are their mutual combinations, such as midpoint and offset.

You might ask, why not make *all* the signs positive in equation (1.27)? The reason is that in that case waves would not move in a positive direction along the space axes. This would be especially unnatural when the space axis was a radius. Atoms, like geophysical sources, always radiate from a point to infinity, not the other way around. Thus, in equation (1.27) the sign of the spatial frequencies must be opposite that of the temporal frequency.

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

1.5.3 Simple examples of 2-D FT

An example of a **two-dimensional Fourier transform** of a pulse is shown in Figure 1.5. Notice the location of the pulse. It is closer to the time axis than the frequency axis. This will affect the real part of the FT in a certain way (see exercises). Notice the broadening of the pulse. It was an impulse smoothed over time (vertically) by convolution with (1,1) and over space (horizontally) with (1,4,6,4,1). This will affect the real part of the FT in another way.

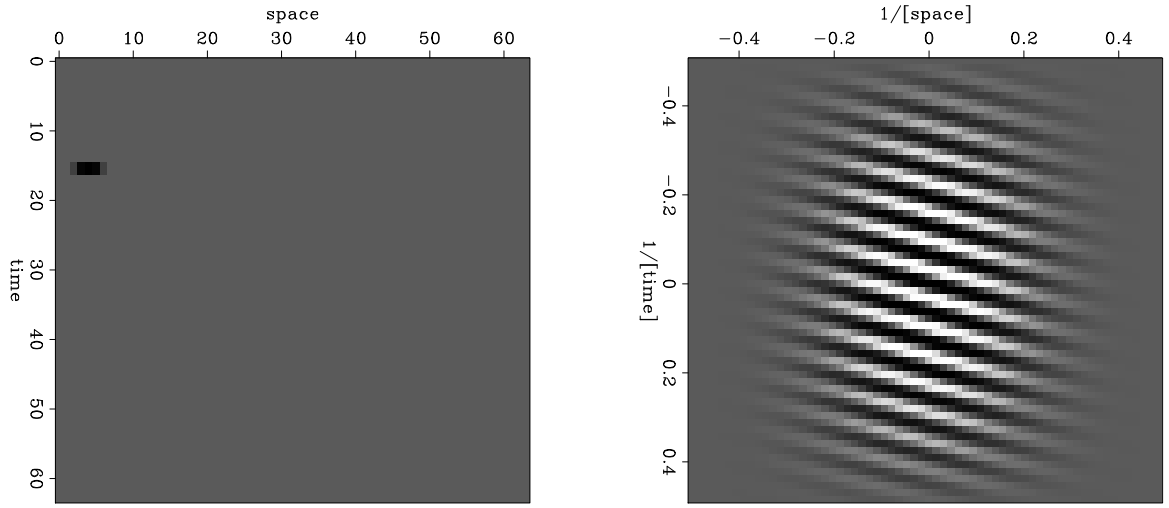

Figure 1.5: A broadened pulse (left) and the real part of its FT (right). ft1-ft2dofpulse [ER]

Another example of a two-dimensional Fourier transform is given in Figure 1.6. This example simulates an impulsive air wave originating at a point on the x-axis. We see a wave propagating in each direction from the location of the source of the wave. In Fourier space there are also two lines, one for each wave. Notice that there are other lines which do not go through the origin; these lines are called "**spatial alias**es." Each actually goes through the origin of another square plane that is not shown, but which we can imagine alongside the one shown. These other planes are periodic replicas of the one shown.

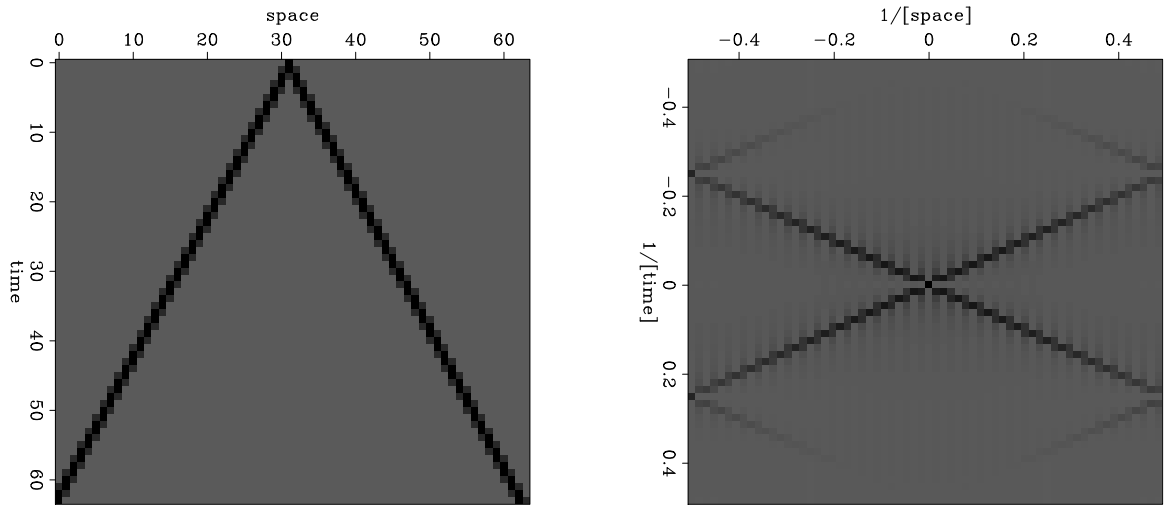

Figure 1.6: A simulated air wave (left) and the amplitude of its FT (right). ft1-airwave [ER]

EXERCISES:

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

2 What would change in Figure 1.5 if the pulse were moved (a) earlier on the t-axis, and (b) further on the x-axis? What would change in Figure 1.5 if instead the time axis were smoothed with (1,4,6,4,1) and the space axis with (1,1)?

3 What would Figure 1.6 look like on an earth with half the earth velocity?

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

5 Explain the horizontal "layering" in Figure 1.4 in the plot of $P(\omega,x)$. What determines the "layer" separation? What determines the "layer" slope?

1.6 THE HALF-ORDER DERIVATIVE WAVEFORM

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

$$t^{n-1}\,\text{step}(t) \quad = \quad \text{FT}\left(\frac{1}{(-i\omega)^n}\right) \tag{1.28}$$

Proof of the validity of equation (1.28) for integer values of n is by repeated indefinite integration which also indicates the need of an $n!$ scaling factor. Proof of the validity of equation (1.28) for fractional values of n would take us far afield mathematically. Fractional values of n, however, are exactly what we need to interpret Huygen's secondary wave sources in 2-D. The factorial function of n in the scaling factor becomes a gamma function. The poles suggest that a more thorough mathematical study of convergence is warranted, but this is not the place for it.

The principal artifact of the hyperbola-sum method of 2-D migration is the waveform represented by equation (1.28) when $n = 1/2$. For $n = 1/2$, ignoring the scale factor, equation (1.28) becomes

$$\frac{1}{\sqrt{t}}\,\text{step}(t) \quad = \quad \text{FT}\left(\frac{1}{\sqrt{-i\omega}}\right) \tag{1.29}$$

A waveform that should come out to be an impulse actually comes out to be equation (1.29) because Kirchhoff migration needs a little more than summing or spreading on a hyperbola.

To compensate for the erroneous filter response of equation (1.29) we need its inverse filter. We need $\sqrt{-i\omega}$. To see what $\sqrt{-i\omega}$ is in the time domain, we first recall that

$$\frac{d}{dt} \quad = \quad \text{FT } (-i\omega) \tag{1.30}$$

A product in the frequency domain corresponds to a convolution in the time domain. A time derivative is like convolution with a doublet $(1,-1)/\Delta t$. Thus, from equation (1.29) and equation (1.30) we obtain

$$\frac{d}{dt}\,\frac{1}{\sqrt{t}}\,\text{step}(t) \quad = \quad \text{FT } \left(\sqrt{-i\omega}\right) \tag{1.31}$$

Thus, we will see the way to overcome the principal artifact of hyperbola summation is to apply the filter of equation (1.31). In chapter 7 we will learn more exact methods of migration. There we will observe that an impulse in the earth creates not a hyperbola with an impulsive waveform but in two dimensions, a hyperbola with the waveform of equation (1.31), and in three dimensions, a hyperbola of revolution (umbrella?) carrying a time-derivative waveform.

1.6.1 Hankel tail

The waveform in equation (1.31) often arises in practice (as the 2-D Huygens wavelet). Because of the discontinuities on the left side of equation (1.31), it is not easy to visualize. Thinking again of the time derivative as a convolution with the doublet $(1,-1)/\Delta t$, we imagine the 2-D Huygen's wavelet as a positive impulse followed by negative signal decaying as $-t^{-3/2}$. This decaying signal is sometimes called the "**Hankel tail**." In the frequency domain $-i\omega = |\omega| e^{-i90^\circ}$ has a 90 degree phase angle and $\sqrt{-i\omega} = |\omega|^{1/2} e^{-i45^\circ}$ has a **45 degree phase angle**.

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

```
        if( adj == 0)   { yy(i) = yy(i) + cv(i)}
        else            { xx(i) = xx(i) + cv(i)}
return; end
```

In practice, it is easiest to represent and to apply the 2-D Huygen's wavelet in the frequency domain. Subroutine `halfdifa()` on the page before is provided for that purpose. Instead of using $\sqrt{-i\omega}$ which has a discontinuity at the Nyquist frequency and a noncausal time function, I use the square root of a causal representation of a finite difference, i.e. $\sqrt{1-Z}$, which is well behaved at the Nyquist frequency and has the advantage that the modeling operator is causal (vanishes when $t < t_0$). Fourier transform is done using subroutine `ftu()` on page 11. Passing an impulse function into subroutine `halfdifa()` gives the response seen in Figure 1.7.

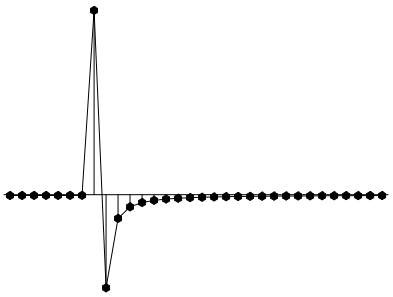

Figure 1.7: Impulse response (delayed) of finite difference operator of half order. Twice applying this filter is equivalent to once applying $(1,-1)$. ft1-hankel [ER]

1.7 References

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

Index

45 degree phase angle, 19

autocorrelation, 7, 8

basement rock, 14

Cauchy function, 10
comb, 10

double-sided exponential, 10

exponential, 10
exponential
    double-sided, 10

fast Fourier transform, 11
Fortran, 13
Fourier sum, 2
Fourier transform
    discrete, 1
    fast, 11
    inverse, 11
    two-dimensional, 13, 14, 16
ft1axis subroutine, 13
ft2axis subroutine, 13
fth subroutine, 12
ftu subroutine, 11

Gaussian, 10
Gibbs sidelobes, 8

halfdifa subroutine, 19
Hankel tail, 19
Hertz, 5

inverse Fourier transform, 11

mesh, 5

negative frequency, 7
Nyquist frequency, 3

pad2 subroutine, 10

quefrency, 10

random, 10
rectangle function, 8
resolution, 5

scale factor, 6
scale subroutine, 6
sign convention, 16
simpleft subroutine, 6
sinc, 8
spatial alias, 14, 17
spectrum, 7
subroutine
    `ft1axis`, FT 1-axis, 13
    `ft2axis`, FT 2-axis, 13
    `fth`, FT, Hale style, 12
    `ftu`, unitary FT, 11
    `halfdifa`, half derivative, 19
    `pad2`, round up to power of two, 10
    `scale`, scale an array, 6
    `simpleft`, slow FT, 6

zero pad, 10