IMPLEMENTING f-k MIGRATION AND DIFFRACTION

Walt Lynn

INTRODUCTION

The previous paper of this section reviewed the concept of migrating seismic data in the frequency domain (an idea presented by Stolt [1]). This article will present the practical side of implementing frequency (f-k) domain migration. It is not intended to be a comparison between the time and frequency domain algorithms, but rather a description of our implementation of the method. We will also investigate the inverse of f-k migration, f-k diffraction, which can be considered as a frequency domain operation that produces synthetic seismograms for an arbitrary constant-velocity model.

The implementation of the f-k methods involves a straightforward mapping, requiring interpolation and scaling, of one frequency domain to another. After discussing these frequency domain mappings for migration and diffraction, we will briefly examine an impulse response for each case.

The interpolation of the complex frequency domain values can cause spurious results superimposed on the output if not done correctly. Two different interpolation schemes, one arithmetic and one geometric, are presented to demonstrate this problem. One result, using the superior geometric interpolation, is compared with an exact solution to test the accuracy of the interpolation and is shown to be in close agreement.

Lastly, the use of f-k diffraction to generate synthetic seismograms is demonstrated with a simple example. An appendix is included containing the algorithms used for f-k diffraction and migration.

In this paper, small letters are used to denote space and time functions, and capital letters their Fourier transforms. So $p(t,x)$ transforms to $P(\omega,k_x)$ and $p(z,x)$ transforms to $P(k_z,k_x)$.

PERIODICITY

The f-k method has advantages in both its simplicity and the computational advantages afforded it by high-speed FFT algorithms. Stolt [1] points out that the additional advantage of more accurate derivatives (a scaling by frequency in the frequency domain) is obtained only if the input is adequately sampled in both space and time.

There are two major limitations on the f-k method, however: (1) there is no freedom to choose boundary conditions as the use of discrete Fourier transforms (DFTs) inherently assumes periodic boundary conditions, and (2) the velocity must be constant. To illustrate the problem associated with the first, consider the migration of a zero-offset section consisting of a single non-zero arrival at $t = t_0$, $x = x_0$ [Fig. (1a)], i.e.,

$$p(t,x,z=0) = \delta(t - t_0)\, \delta(x - x_0)\, w(t).$$

If X and T are the dimensions of the grid, then the periodic boundary conditions imply an arrival at

$$x = x_0 \pm nX, \qquad n = 0, \pm 1, \pm 2, \ldots ,$$

$$t = t_0 \pm mT, \qquad m = 0, \pm 1, \pm 2, \ldots .$$

This is shown in Fig. 1(b). The correct migration consists, of course, of a semicircle with a bottom at $z_0 = v\, t_0$ [the heavy line in Fig. 1(c)]. In actuality, however, the output will be the migration of all of the implied arrivals. Some of these periodic semicircles that pass through the output grid are shown in Fig. 1(d). In practice, these are very weak relative to the correct structure, but of course can be made arbitrarily small by padding zeros to the sides and the bottom of the grid.

There is no apparent way to handle spatially varying velocity in the frequency domain.

MAPPING OF THE FREQUENCY DOMAINS

In the frequency domain, migration and diffraction are simply mappings of one domain to another; that is, for migration we map $P(\omega,k_x)$ into $P(k_z,k_x)$ and vice-versa for diffraction. As we are working with discretely sampled data

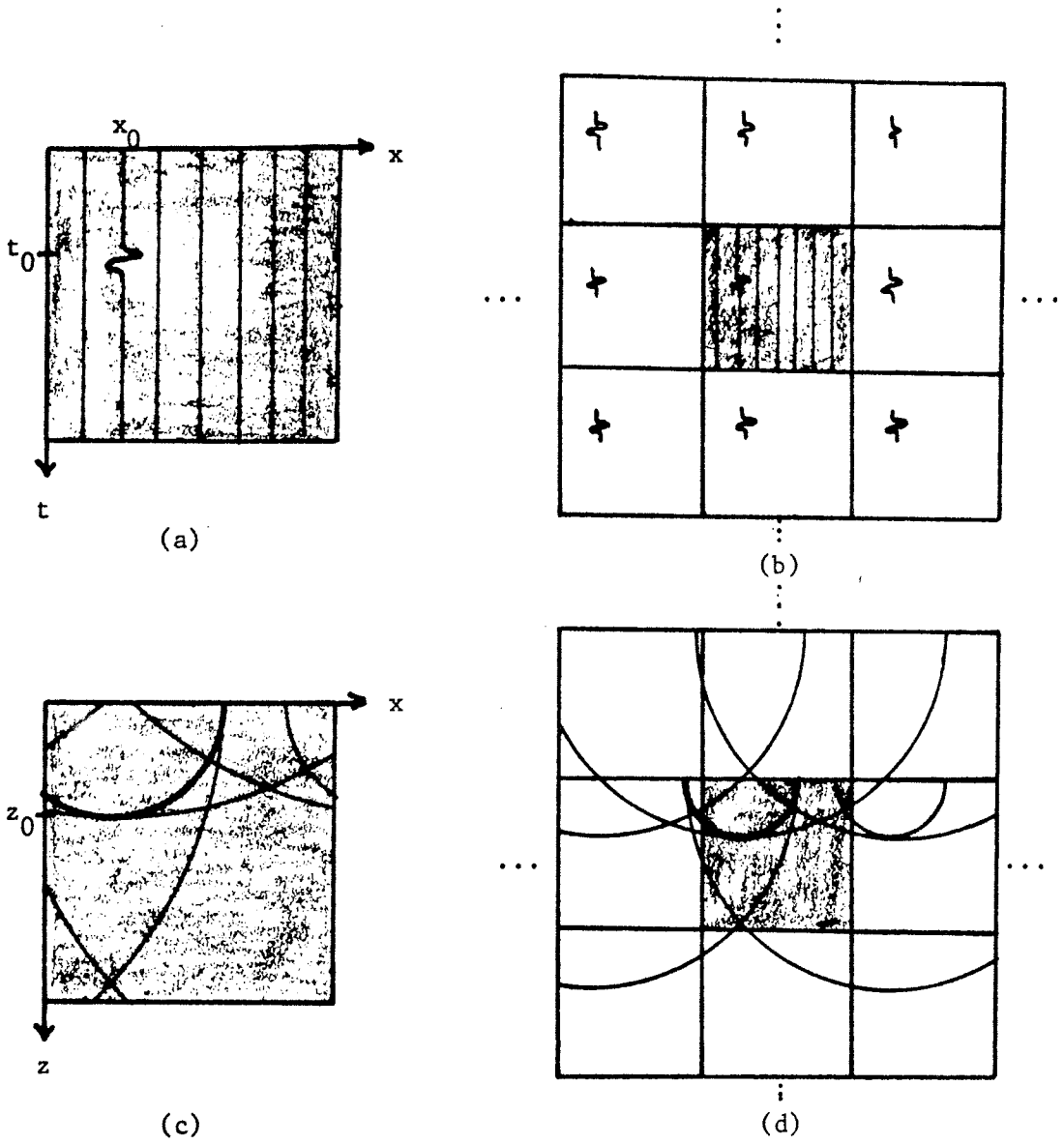

FIGURE 1.—Inherent periodicity implied by the use of discrete Fourier transforms. (a) The desired input, $p(t,x)$, which here is $\delta(t - t_0)\delta(x - x_0)w(t)$. (b) The input implied by the periodic boundary conditions of the DFT. The dots imply a continuation *ad infinitum*. (c) The migrated section. The correct structure is shown by the heavy line. (d) The origin of the spurious structures in (c). The spurious structures are not as troublesome as they may seem since they are substantially weaker than the correct one. They may, of course, be attenuated to any desired level simply by padding the boundaries of the original input with zeros.

sets of finite length, there will be some data-dependent boundaries on the individual domains, and because of this, parts of one domain may map outside the other.

In the migration problem we first Fourier transform the seismic gather $p(t,x)$ to obtain $P(\omega,k_x)$. This information lies on a grid in (ω,k_x) space between limits of $\pm\omega_N$ and $\pm k_{xN}$, where the subscript-N refers to the respective Nyquist frequency (see Fig. 2). Migration is done by changing variables from ω to k_z,

$$\omega = -v k_z [1 + (k_x/k_z)^2]^{1/2} , \tag{1a}$$

or

$$k_z = -\omega/v \, [1 - (vk_x/\omega)^2]^{1/2} . \tag{1b}$$

The minus sign indicates that we are considering upgoing waves.

The shaded region in Fig. 2(a) shows the migration mapping of the known (ω,k_x) domain to the desired (k_z,k_x) domain. To do the inverse DFT, and thus get $p(z,x)$, we need the values of $P(k_z,k_x)$ at specific grid points in the (k_z,k_x) domain [given by the rectangular area on the right in Fig. 2(a)]. Note that the minus sign in the transformation maps the first quadrant into the fourth and the second into the third.

There are two practical problems. One is that the mapping does not fill up the desired rectangular grid in the (k_z,k_x) domain. The second is that, with the exception of $k_x = 0$, the mapped points do not lie on grid points in the (k_z,k_x) domain. To see how these problems are handled it is useful to consider the inverse mapping from the desired (k_z,k_x) space to (ω,k_x) space.

We want to create a grid of uniformly-spaced points in the (k_z,k_x) plane. To do this we fix k_x and scan through the appropriate values of k_z. If $d\omega$ is the frequency interval in the known (ω,k_x) domain, then k_z ranges from $-\omega_N/v$ to $+\omega_N/v$ with $k_z = \omega/v$. The desired (k_z,k_x) point is mapped back to (ω,k_x) space using Eq. (1a). If it lies outside the rectangular grid in Fig. 2(a), $P(k_z,k_x)$ is set equal to zero. If (k_z,k_x) maps to a point (ω,k_x) within the known region, then $P(k_z,k_x)$ is computed by interpolating between the values at known grid points $P(\omega_1,k_x)$ and $P(\omega_2,k_x)$, where $\omega_1 \leq \omega \leq \omega_2$. The interpolation will be discussed in greater detail later.

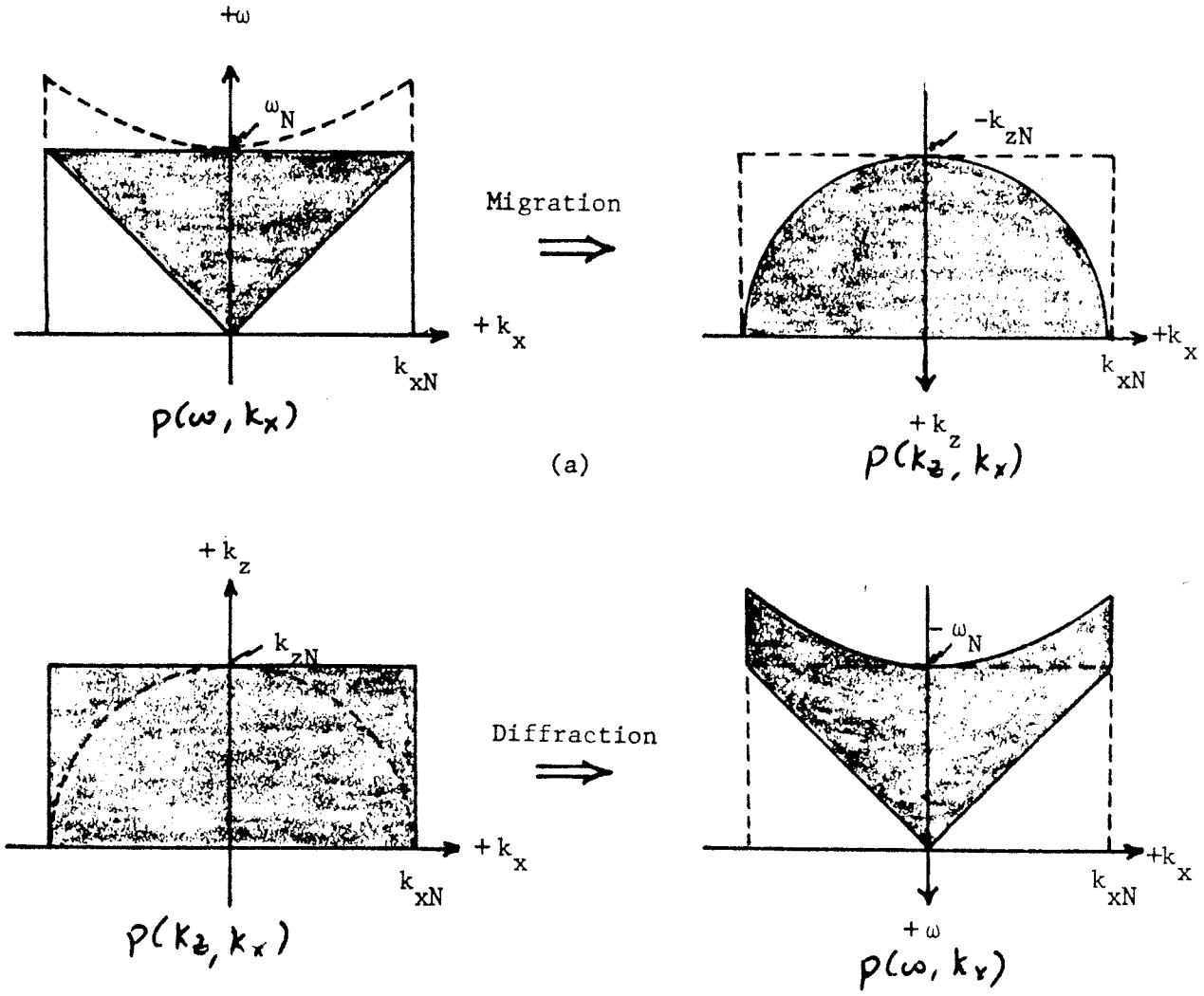

FIGURE 2.—Frequency domain mappings for (a) migration: $P(\omega,k_x)$ to $P(k_z,k_x)$, and (b) diffraction: $P(k_z,k_x)$ to $P(\omega,k_x)$. Note that only half of the ω and k_z domains are shown and that $+\omega$ maps into $-k_z$ and vice versa. The left-hand side domains are the 2-D Fourier transforms of the input data [$(p(t,x)$ in (a) and $p(z,x)$ in (b)] and are known at discrete grid points extending out to their respective Nyquist frequencies (shaded areas). The Nyquist frequencies here are all equal to π and the velocity equals 1. (See Figs. 3 and 5 for a slightly different case.) To accomplish the migration or diffraction, $P(k_z,k_x)$ or $P(\omega,k_x)$ must be found at discrete grid points in the unknown domains on the right. Since the desired domains (bounded by the dashed lines) and the known domains (shaded areas) do not coincide exactly, an interpolation must be done to evaluate the values at the desired points. The unknown function values are found by mapping the grid points back into the known domain and interpolating between the two closest known function values. Where a desired grid point maps outside the known domain (i.e., beyond the Nyquist frequency), the value of the unknown function is set to zero. Interpolation is done by a weighted average of the complex logarithms of the known function (see Fig. 7). In the (ω,k_x) domains, the areas below the diagonals and above the k_x-axis represents evanescent energy, i.e., where $|k_x| > |\omega/v|$.

The diffraction problem is simply the inverse of the migration problem. We start with $P(k_z,k_x)$, which comes from Fourier transforming the initial source distribution $p(z,x)$. The mapping from the known to unknown domains is shown in Fig. 2(b). We want $P(\omega,k_x)$ at uniformly-spaced grid points, where $\Delta\omega = v\,\Delta k_z$. To do this we again fix k_x, scan the desired ω's and find the corresponding k_z using (1b). A negative discriminant in (1b) implies evanescent waves, and for these values of (ω,k_x), $P(\omega,k_x)$ is set to zero. For positive discriminants, $P(\omega,k_x)$ is found by interpolating between $P(k_{z1},k_x)$ and $P(k_{z2},k_x)$, where $k_{z1} \le k_z \le k_{z2}$. Before considering the problems associated with sampling and with interpolating the complex frequency domain values, we will first examine an impulse response for both the diffraction and migration filters.

DIFFRACTION OF A POINT SCATTERER

Figure 3(a) shows the input model $p(z,x) = \delta(z-z_0)\,\delta(x-x_0)$ used to examine an impulse response of the diffraction filter. The parameters used to generate the model are given in the figure caption. Since we must assume a constant velocity medium, we expect the result to be a hyperbola in (t,x) space. The first step is to Fourier transform $p(z,x)$ to $P(k_z,k_x)$. For the input in Fig. 3(a), this is

$$P(k_z,k_x) = e^{ik_x x_0 + ik_z z_0}. \qquad (2)$$

The real and imaginary parts of this transform are shown in Fig. 3(b). $P(\omega,k_x)$, Fig. 3(c), is found by estimating $P(-[(\omega/v)^2 - k_x^2]^{1/2},k_x)$ using Eq. (1b). The final result $p(t,x)$ is found by inverse-Fourier-transforming $P(\omega,k_x)$ [Fig. 3(d)]. The extraneous hyperbolic tails are caused by the periodicity inherent in the method. This is clearly shown in Fig. 4 where $p(t,x)$ has been plotted side by side several times.

MIGRATION FILTER IMPULSE RESPONSE

Figure 5(a) shows the input function $p(t,x)$, and 5(b) its 2-D Fourier transform $P(\omega,k_x)$, used to show one impulse response of the migration filter. The input is the same as in the diffraction case, but now is considered a function of x and t. $P(k_z,k_x)$ is found by estimating $P(-v[k_x^2+k_z^2]^{1/2},k_x)$ and is shown in Fig. 5(c). The final migrated result $p(z,x)$ is the inverse 2-D Fourier transform of 5(c) and is shown in Fig. 5(d).

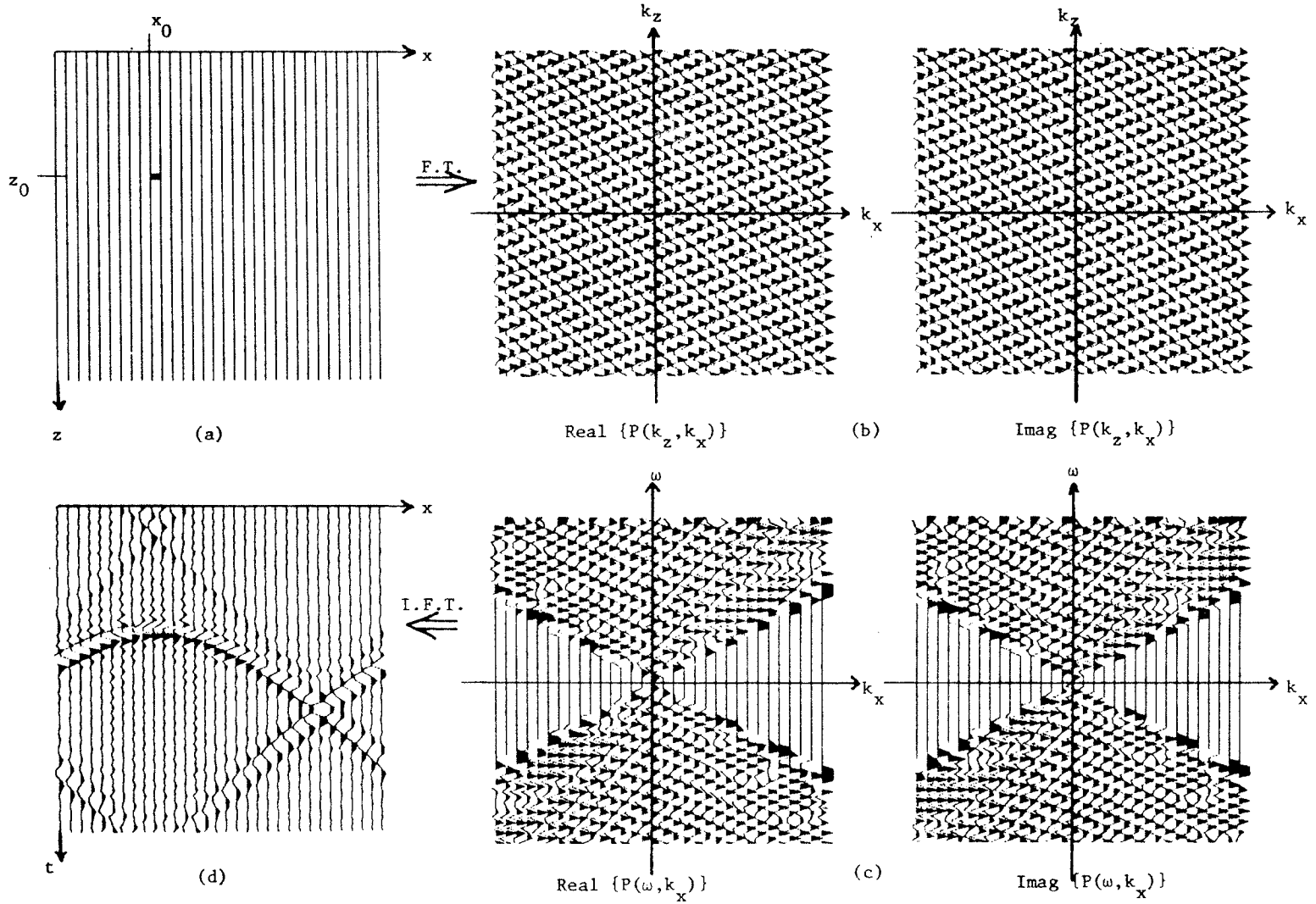

FIGURE 3.—Diffraction of a point scatterer. Input model parameters are $dz = dx = 1$, velocity = 1, number of time points $nt = 64$, and number of traces $nx = 32$. Nyquist frequencies are $k_{xN} = k_{zN} = \omega_N = \pi$. Counterclockwise from upper left: (a) Input model, $p(z,x) = \delta(z - z_0)\delta(x - x_0)$. (b) Real and imaginary parts of the 2-D DFT of $p(z,x)$, $P(k_z,k_x)$. (c) Real and imaginary parts of $P(\omega,k_x)$ obtained from (b) using Eq. (1b). Note the Hermetian property of the transforms due to the pure real input data, i.e., $P(\omega,k_x) = P^*(-\omega,-k_x)$. (d) The final result $p(t,x)$. See Fig. 4 for the origin of the extraneous hyperbolic tails.

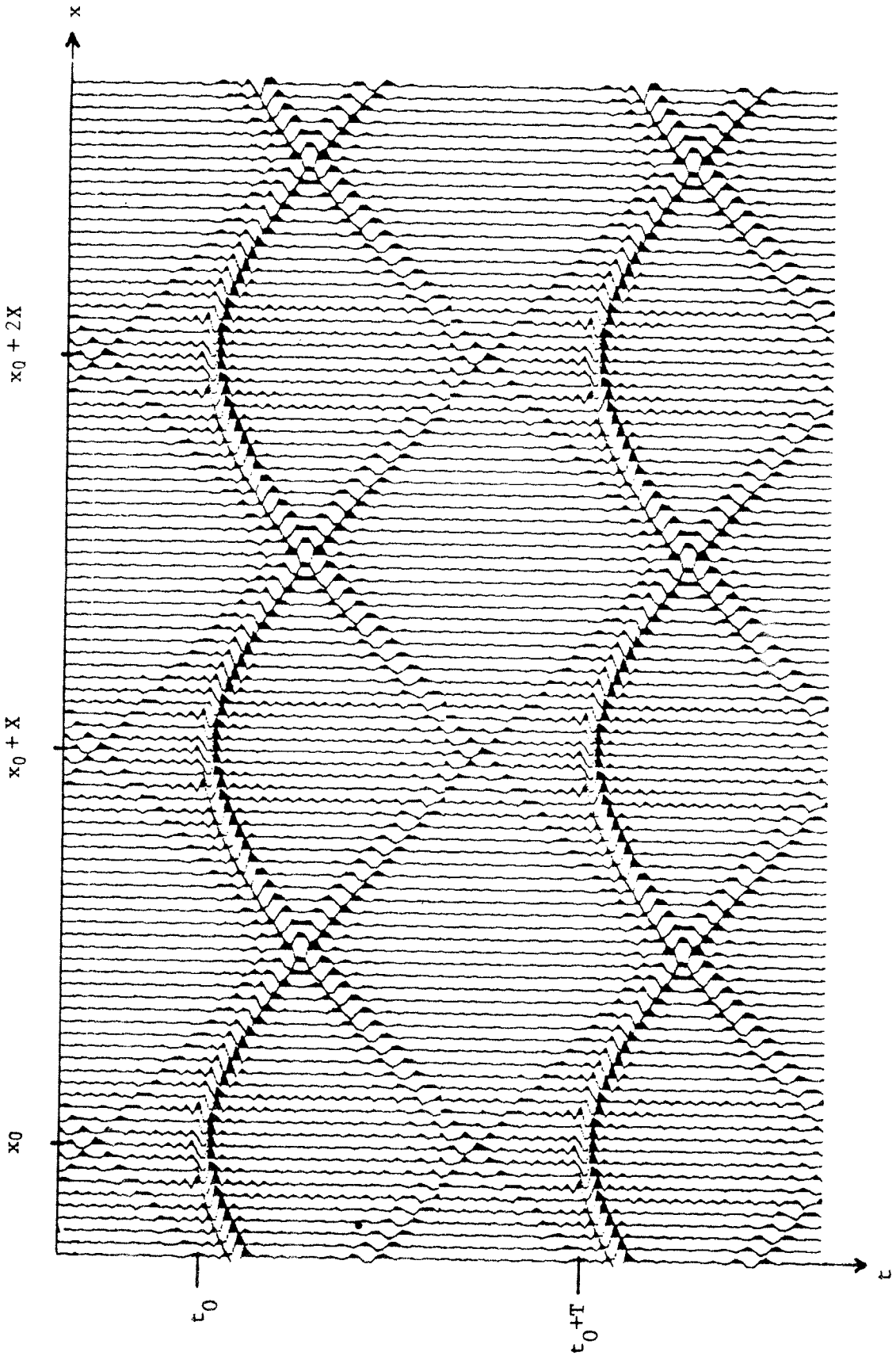

FIGURE 4.—The output from the diffraction example in Fig. 3 plotted several times side by side to demonstrate the periodicity of the f-k diffraction and the origin of the extraneous hyperbolic tails.

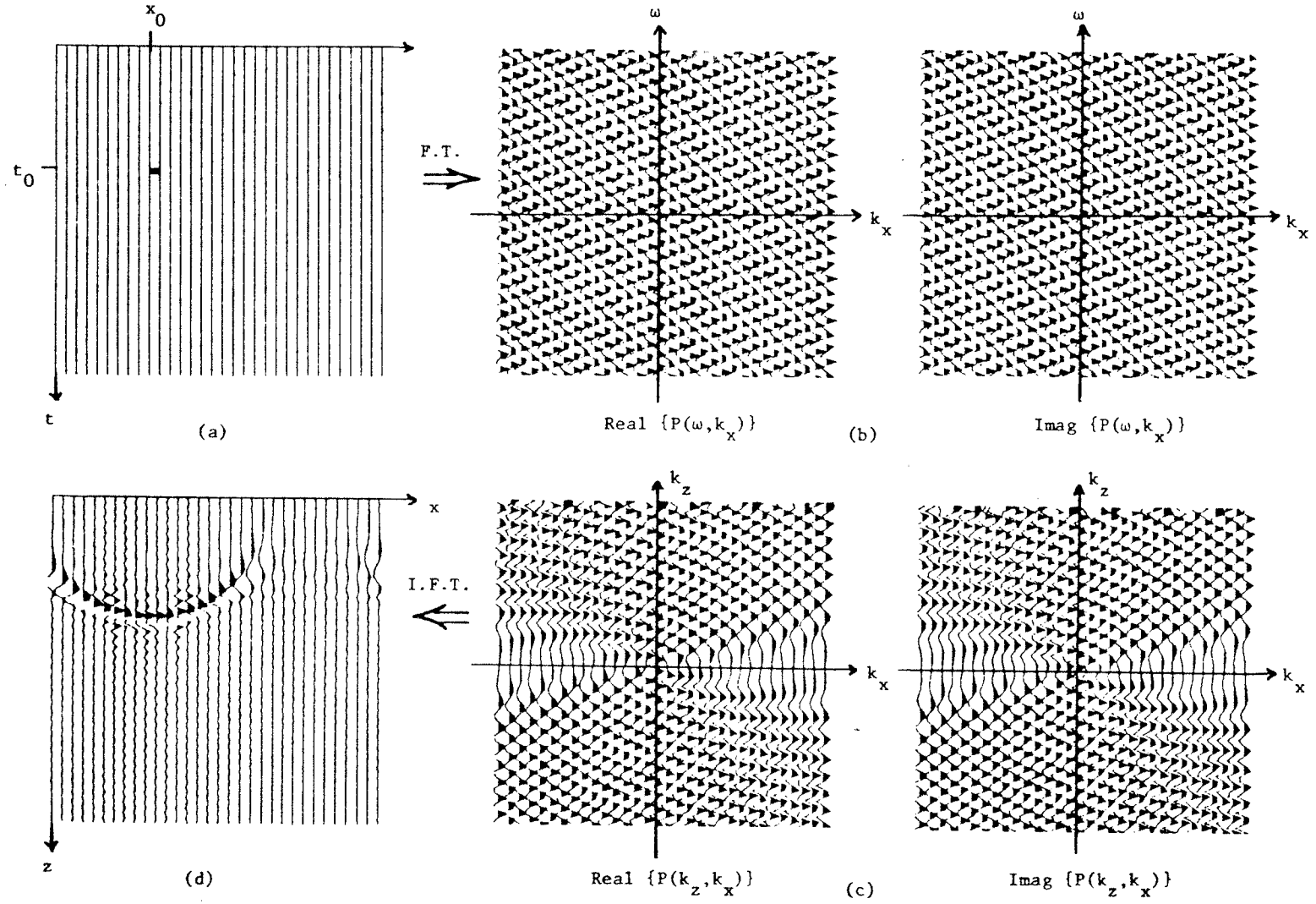

FIGURE 5.—Migration filter impulse response. Model parameters are the same as in Fig. 3. Counterclockwise from upper left: (a) Input model, $p(t,x) = \delta(t - t_0)\delta(x - x_0)$. (b) Real and imaginary parts of the Fourier transform of $p(t,x)$, $P(\omega,k_x)$. (c) $P(k_z,k_x)$ obtained from (b) using Eq. (1a). (d) The migrated result $p(z,x)$.

INTERPOLATION

A word about interpolating complex numbers is in order. In our first attempts to test the f-k migration method we kept obtaining spurious structures along with the correct ones. For every correct semicircle, there was an incorrect one, upside-down and tangent to the bottom of the correct one. Figures 6(b) and 6(e) show this for two different inputs, one with a spike at $0.4\ t_{max}$ and the other with a spike at $0.5\ t_{max}$. The problem was traced to the interpolation scheme. In Figs. 6(b) and (e), a linear interpolation was used, and in Figs. 6(c) and (f), a geometric interpolation was used. Clearly, the latter is superior, and Fig. 7 demonstrates why.

Figure 7(a) shows two adjacent values of $P(\omega,k_x)$ at $\omega = \omega_1, \omega_2$. We want to estimate $P(\omega_1+\Delta\omega',k_x)$. Let q be the fraction of the interval that $\Delta\omega'$ is from ω_1, i.e., $q = \Delta\omega'/(\omega_2 - \omega_1)$. One method of estimating $P(\omega_1+\Delta\omega',k_x)$ is to use a simple linear interpolation:

$$P(\omega_1+\Delta\omega',k_x) = (1-q)\ P(\omega_1,k_x) + q\ P(\omega_2,k_x)\ .$$

An alternate scheme is to use a geometric interpolation between $P(\omega_1)$ and $P(\omega_2)$, i.e.,

$$P(\omega_1+\Delta\omega') = P(\omega_1)[P(\omega_2)/P(\omega_1)]^q\ , \tag{3}$$

or

$$\ell n\ P(\omega_1+\Delta\omega') = (1-q)\ \ell n\ P(\omega_1) + q\ \ell n\ P(\omega_2)\ . \tag{4}$$

Since the imaginary part of the complex logarithm of P is its phase, then Eq. (4) represents a linear interpolation of the phases and a geometric average of the amplitudes.

In Fig. 7(a), $P(\omega_1,k_x)$ and $P(\omega_2,k_x)$ are separated by ~87° and in Fig. 7(b) by ~219°. As an example, let $q = 0.2$, meaning we want to estimate $P(\omega_1+0.2\Delta\omega,k_x)$, where $\Delta\omega = \omega_2 - \omega_1$. Observe that in Fig. 7(a), the phase of the linearly interpolated value (solid phasor) is approximately correct, whereas in 7(b) the phase is in severe error. When the phase is only a little bit in error, a weak spurious semicircle results as in Fig. 6(b). On the other hand, when $P(\omega,k_x)$ is rapidly oscillating, as in Fig. 7(b), strong spurious

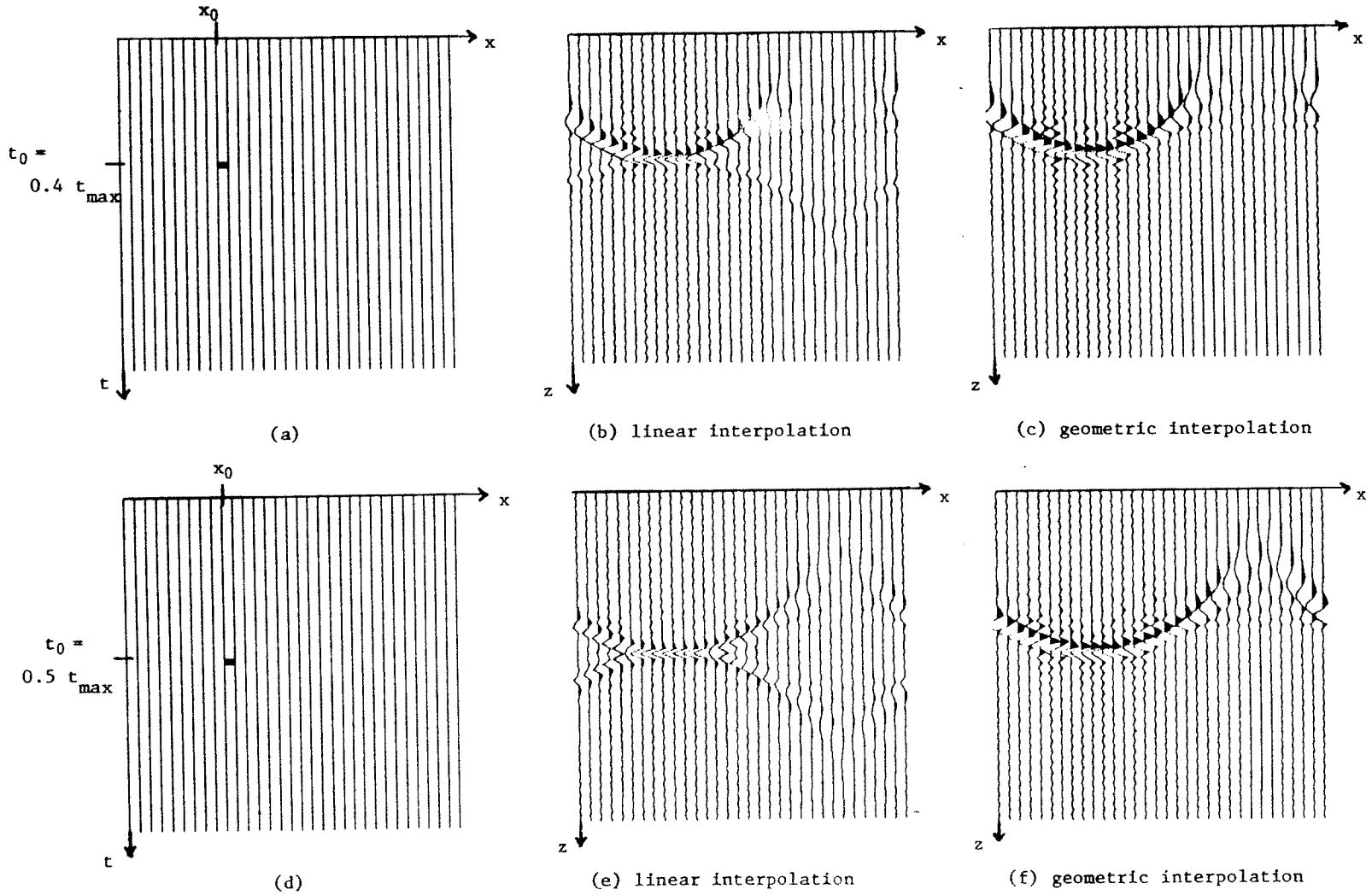

FIGURE 6.—Comparison of interpolation techniques for two different input models. Linear interpolation is done using $P(\omega_1 + \Delta\omega', k_x) = (1 - q)\ P(\omega_1, k_x) + q\ P(\omega_2, k_x)$. Geometric interpolation is done using $P(\omega_1 + \Delta\omega', k_x) = P(\omega_1, k_x)[P(\omega_2, k_x)/P(\omega_1, k_x)]^q$. (a) Input model, $p(t,x)$ with a spike at $t = 0.4\ t_{max}$. (b) Migrated result using linear interpolation. (c) Migrated result using geometric interpolation. (d) Input model, $p(t,x)$ with a spike at $t = 0.5\ t_{max}$. (e) Migrated result using linear interpolation. (f) Migrated result using geometric interpolation. A spike at $0.5\ t_{max}$, produces the fastest oscillation in the frequency domain and hence the equal strengths of the semicircles in (e). The geometric interpolation is clearly superior because it correctly handles the phase (see Fig. 7).

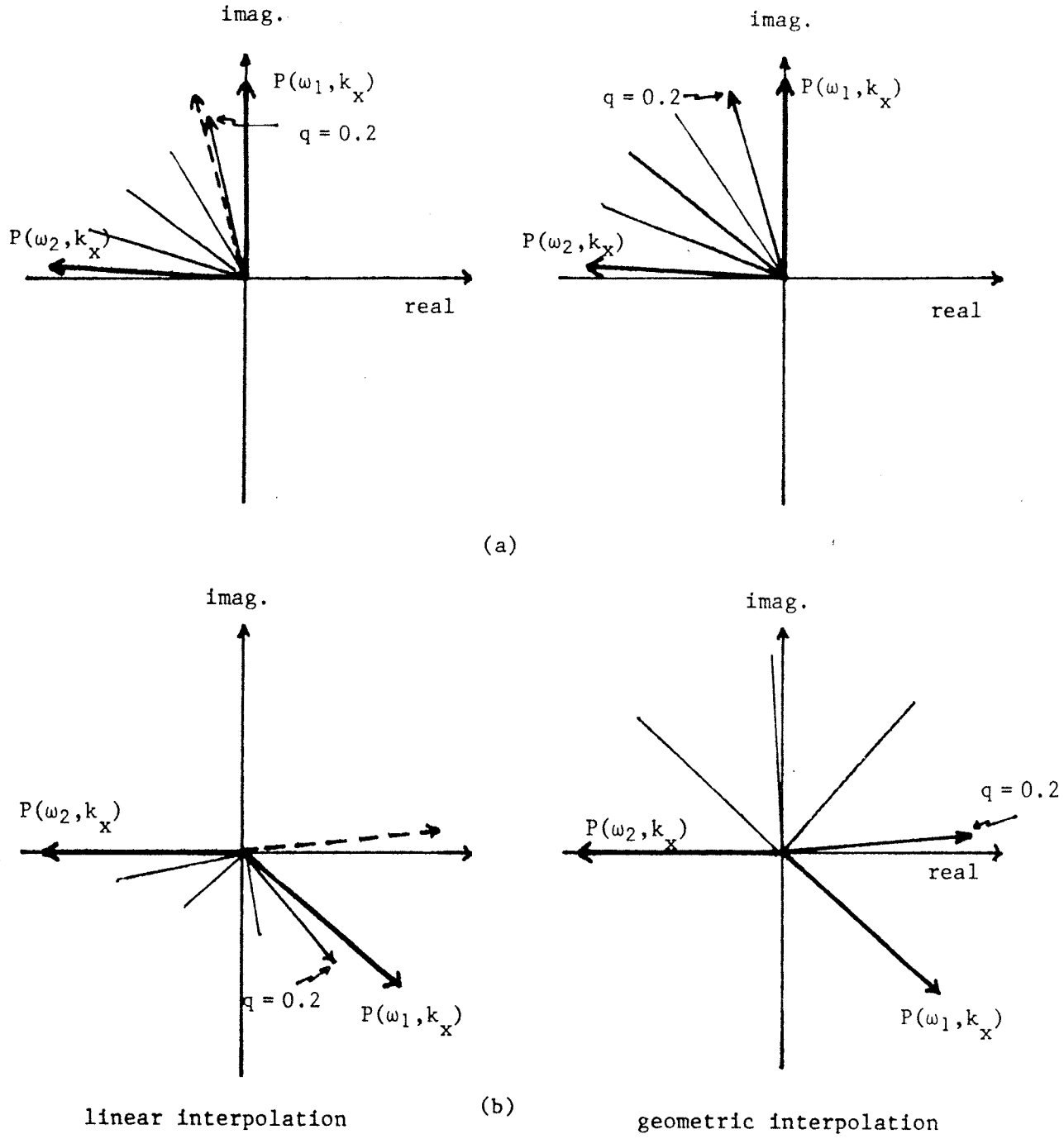

FIGURE 7.—Comparison of linear (left) and geometric (right) interpolation of complex numbers for two different cases. The known values are $P(\omega_1,k_x)$ and $P(\omega_2,k_x)$ (heavy lines). The desired value is $P(\omega',k_x)$ where $\omega' = \omega_1 + 0.2\,\Delta\omega$. In (a) the two known phasors are separated by ~87° and in (b) by ~219°. In both cases the phasors spiral counterclockwise. The linear interpolation ignores this and places $P(\omega',k_x)$ as the arithmetic average of the two known phasors, whereas the geometric interpolation does a linear interpolation of the phase and puts the estimated value in its correct orientation. The correct phasor is shown by the dashed line. The lighter phasors show interpolated values for $q = 0.4$, 0.6, and 0.8.

semicircles will result as in Fig. 6(e). We conclude from this that keeping the phase correct is far more important than keeping the amplitudes correct.

Just how good is the geometric interpolation? For a simple delta-function input, it is a simple matter to compute the exact frequency domain representation of the migrated output. Fourier-transforming the full wave equation with respect to x and z gives

$$P = - v^2(k_x^2 + k_z^2) P ,$$

which has solutions

$$P(t,k_x,k_z) = P(t_0,k_x,k_z)\, e^{\pm i v (k_x^2+k_z^2)^{1/2}} . \qquad (5)$$

The frequency domain representation using Eq. (5) can be compared directly with the interpolated frequency domain in the f-k migration.

Using the input model, $p(t,x)$ of Fig. 5(a), Fig. 8 compares an f-k migration using geometric interpolation with the result of inverse transforming Eq. (5). The differences in the migrated results, Figs. 8(a) and (c), are slight, indicating that the interpolation is adequate. The real and imaginary parts of the exact Fourier transform, Fig. 8(d), are also shown for comparison with the interpolated Fourier transform, Fig. 8(b).

DIFFRACTION EXAMPLE

Finally, we show an example of using the f-k method to generate a synthetic seismogram. We are not advocating using this method over others, but rather only showing how it can be used for constant-velocity synthetics. Figure 9(a) shows a buried focus input model, $p(z,x)$, with no vertical exaggeration, and 9(b) the synthetic seismogram generated using the f-k diffraction method. Note that the method has correctly incorporated a 90° phase shift in the reverse branch of the triplication.

REFERENCE

[1] STOLT, R. H., "Migration in momentum space," *Geophysics* (submitted for publication).

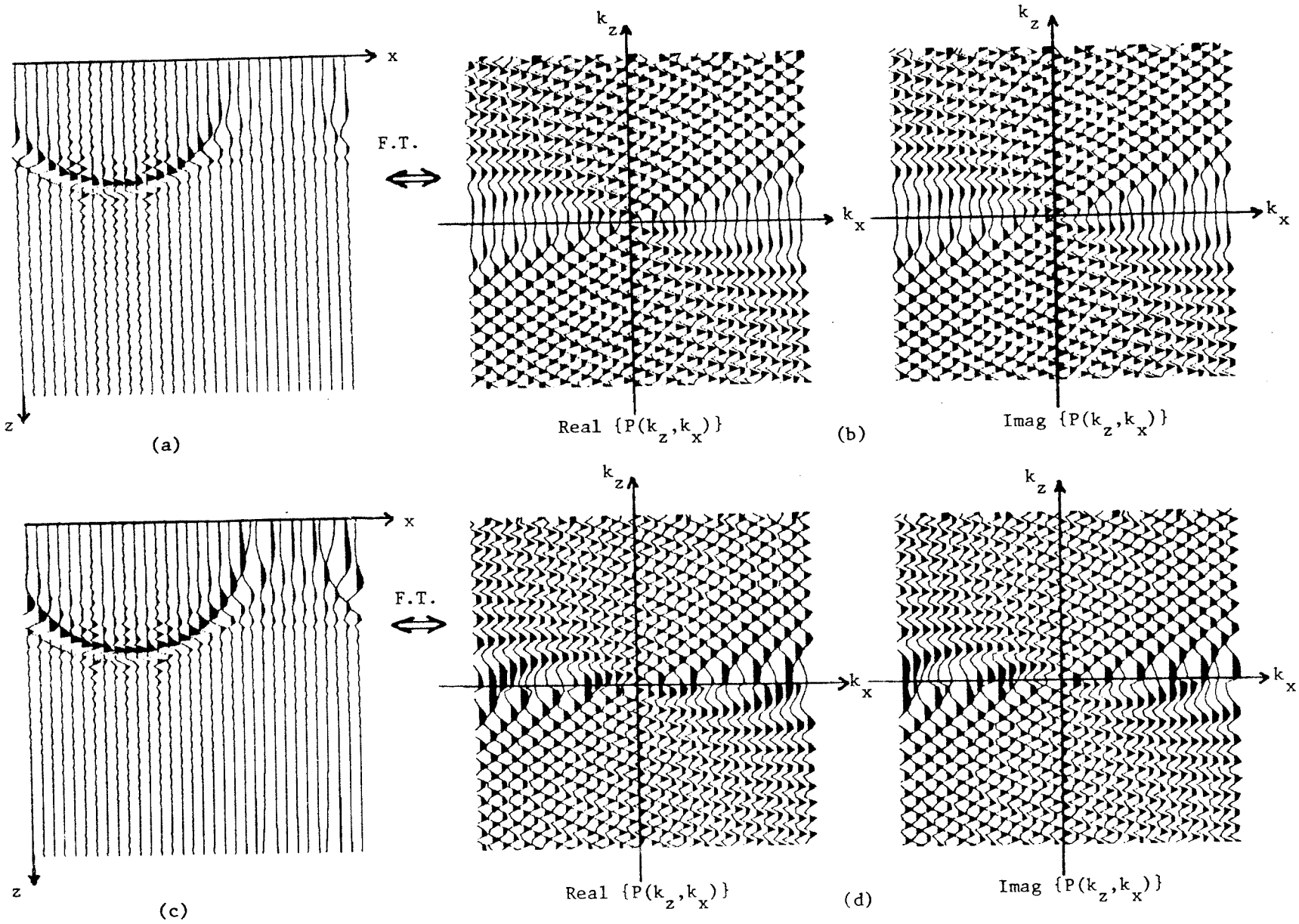

FIGURE 8.—Geometric interpolation vs. exact solution. Input model is shown in Fig. 5(a). Top: (a) f-k migration using a geometric interpolation. (b) The real and imaginary parts of the Fourier transform of (a). Bottom: (c) Exact result obtained by inverse Fourier transforming Eq. (5) shown in (d). The two results are nearly identical except at steep dips indicating that the interpolation is very satisfactory.

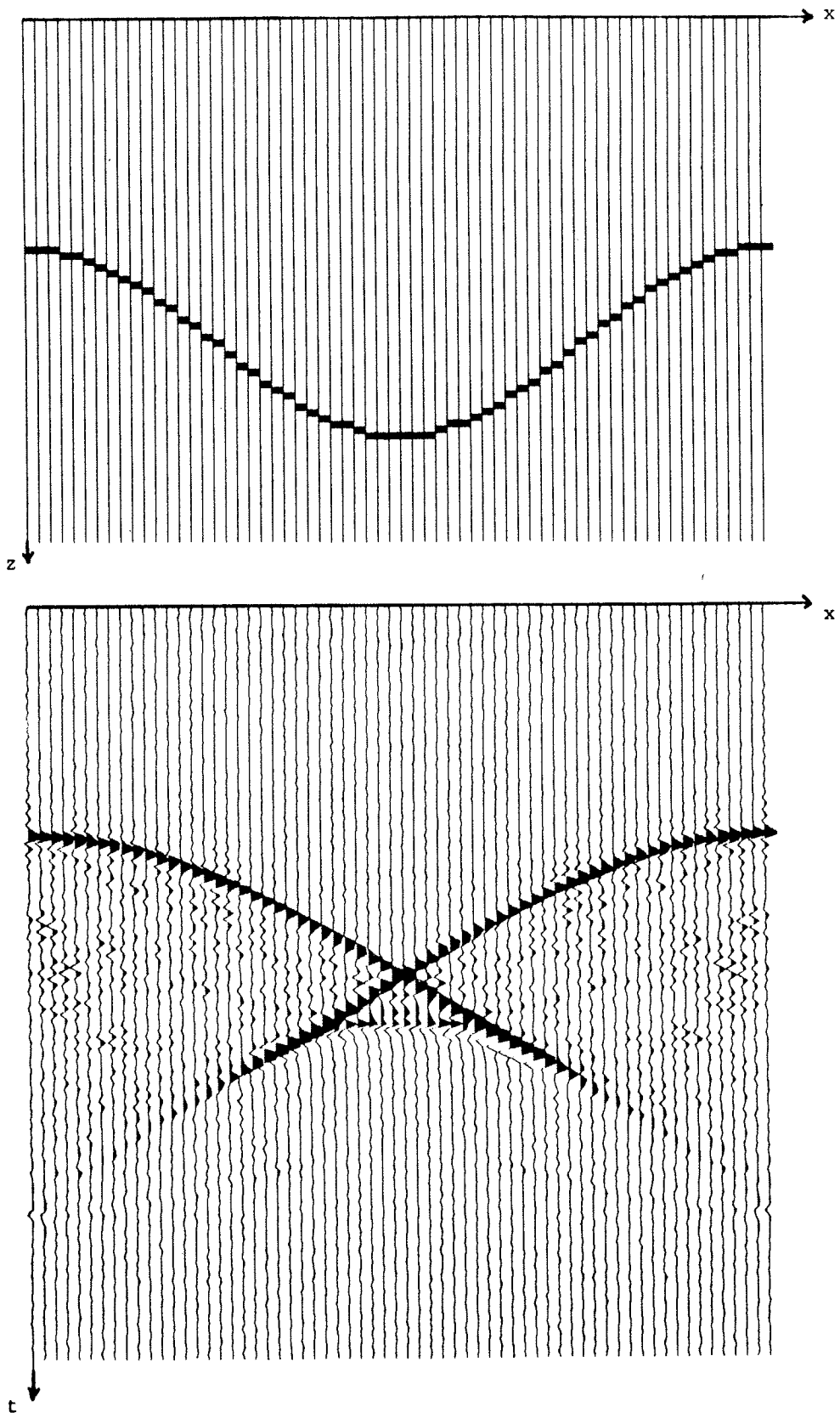

FIGURE 9.—Examples of using f-k diffraction to generate synthetic seismic sections. (a) Input model, p(z,x), plotted with no vertical exaggeration. Maximum dip is about 38°. The grid is 128 × 64 (nz × nx). (b) Diffracted result. Note the triplication due to the buried focus. At the base of the triplication the method has correctly incorporated a 90° phase shift.

APPENDIX

A complete Fortran listing of the f-k migration algorithm is given below followed by a listing of the pertinent changes to be made in it for f-k diffraction. The variable definitions are the same in each program. The input data is a complex 2-D Fourier transform, $P(\omega,k_x)$ or $P(k_z,k_x)$, where the first subscript varies the fastest. Because of core limitations in our computer, we work with only one vector (k_x = constant) at a time. The convention on the DFTs is: if p(t) is a vector of length nt, P(1) = P(ω=0); P(nt/2 + 1) = $P(\omega_{Nyquist})$; and P(nt) = $P(\omega=\omega_{sample} - \Delta\omega) = P(-\Delta\omega)$.

a) Listing of program f-k migrate

```
c       program fkmigrate
c
c       Usage:  fkmigr infile  outfile nt nx
c
c       given:  p(w,kx)  at ikx=1,2,...,nx
c                           iw =1,2,...,nt
c
c       find:   p(kz,kx) at ikx=1,2,...,nx
c                           ikz=1,2,...,nz
c
c       input data, P(w,kx), is complex and stored in strips of kx,
c       starting with kx=0.
c       nx = number of traces (an even power of 2).
c       nt = number of time points (also a power of 2).
c       nxn = subscript of kx nyquist frequency (=nx/2 + 1).
c       ntn = subscript of w (omega) nyquist frequency (= nt/2 + 1).
c       fn = omega nyquist frequency
c       kxn = kx nyquist frequency
c       kzn = kz nyquist frequency
c
        implicit complex (c)
        logical*1 arg(8)
        dimension ci(257),co(257)
        real kxn,kzn,kx,kz
c
c       open input and output files.
c
        call argfil(1,2,512)
        call argfil(2,3,512)
        nbyte = 8
        call getarg(4,arg,nbyte)
        decode(nbyte,100,arg) nt
        call getarg(5,arg,nbyte)
        decode(nbyte,100,arg) nx
100     format(i8)
        write(6,101) nt,nx
101     format(/2x,'frequency domain migration'//2x,
     1          'nt=nz=',i5,2x,'nx=',i4)
c
        ntn=nt/2 + 1
        ntp1=nt+1
        nxn=nx/2 + 1
c
```

```
c       compute nyquist frequencies ,etc.
c       units are meters and seconds.
c
        v=1.
        v2=v*v
        dt=1.
        dx=2.
        pi=3.141592
        pi2=pi*2.
        fn=pi/dt
        df=2*fn/float(nt)
        kxn=pi/dx
        dkx=2*kxn/float(nx)
        kzn=fn/v
        dkz=2*kzn/float(nt)
        dz=pi/kzn
        write(6,104) v,dt,dx,dz
104     format(2x,'v=',f6.0,2x,'dt=',f5.3,2x,'dx=',f5.1,2x,
     1          'dz=',f5.1)
        write(6,105) fn,df,kxn,dkx,kzn,dkz
105     format(2x,'fn =',e9.3,3x,'df=',e9.3/2x,'kxn=',e9.3,
     1          2x,'dkx=',e9.3/2x,'kzn=',e9.3,2x,'dkz=',e9.3)
c
c       This do loop goes from kx=0 to kxnyquist
c       and treats the first and fourth quadrants in fig. 5b.
c
        do 40 jx=1,nxn
          akx=(jx-1)*dkx
          akx2=akx*akx
          read(1) (ci(i),i=1,nt)
          ci(ntp1)=ci(1)
          do 30 jz=1,nxn
            jzc=nt-jz+2
            akz=(jz-1)*dkz
            akz2=akz*akz
            d=sqrt(akx2+akz2)
            f=v*d
            if(f.le.fn) go to 10
            co(jz)=0.
            co(jzc)=0.
            go to 30
10          s=-v
c           avoid dividing by zero.
            if(jx.eq.1 .and. jz.eq.1) go to 11
            s=-v*akz/d
11          r=f/df + 1.
            it=int(r)
            itc=nt-it+2
            itc1=itc-1
            q=r-float(it)
c           interpolate using geometric average
            co(jz)=caveg(q,ci(itc),ci(itc1))
            co(jzc)=caveg(q,ci(it),ci(it+1))
            co(jz)=co(jz)*s
            co(jzc)=co(jzc)*s
30        continue
```

```
40      write(2) (co(i),i=1,nt)
c
c       this do loop goes from kxnyquist+dkx to kxsample-dkx
c       and treats the second and third quadrants in fig. 5b.
c
        nxn1=nxn+1
        do 70 jx=nxn1,nx
          lx=nx-jx+1
          akx=lx*dkx
          akx2=akx*akx
          read(1) (ci(i),i=1,nt)
          ci(ntp1)=ci(1)
          do 60 jz=1,nxn
            jzc=nt-jz+2
            akz=(jz-1)*dkz
            akz2=akz*akz
            d=sqrt(akx2+akz2)
            f=v*d
            if(f.le.fn) go to 50
            co(jz)=0.
            co(jzc)=0.
            go to 60
50          s=-v*akz/d
            r=f/df + 1.
            it=int(r)
            itc=nt-it+2
            itc1=itc-1
            q=r-float(it)
            co(jz)=caveg(q,ci(itc),ci(itc1))
            co(jzc)=caveg(q,ci(it),ci(it+1))
            co(jz)=co(jz)*s
            co(jzc)=co(jzc)*s
60        continue
70      write(2) (co(i),i=1,nt)
        write(6,106)
106     format(2x'fkmigrate finished'/)
        stop
        end
c
        function caveg(q,ca,cb)
c
c       geometric interpolation
c
        implicit complex(c)
        real cabs
        if(cabs(ca).ne.0.) go to 1
        caveg=q*cb
        return
1       if(cabs(cb).ne.0.) go to 2
        caveg=(1.-q)*ca
        return
2       caveg=ca*cexp(q*clog(cb/ca))
        return
        end
```

b) Major definitions and do loops of program f-k diffract

```
c
c       compute nyquist frequencies ,etc.
c       units are meters and seconds.
c
        v=1.
        v2=v*v
        dt=1.
        dx=2.
        pi=3.14159
        pi2=2.*pi
        fn=pi/dt
        df=2*fn/float(nt)
        kxn=pi/dx
        dkx=2*kxn/float(nx)
        kzn=fn/v
        dkz=2*kzn/float(nt)
        dz=pi/kzn
        write(6,104) v,dt,dx,dz
104     format(2x,'v=',f6.0,2x,'dt=',f5.3,2x,'dx=',f5.1,2x,
     1          'dz=',f5.1)
        write(6,105) fn,df,kxn,dkx,kzn,dkz
105     format(2x,'fn =',e9.3,3x,'df=',e9.3/2x,'kxn=',e9.3,
     1          2x,'dkx=',e9.3/2x,'kzn=',e9.3,2x,'dkz=',e9.3)
c
c       this do loop goes from kx=0 to kxnyquist and treats
c       the first and fourth quadrants in figs. 3b.
c
        do 40 jx=1,nxn
          akx=(jx-1)*dkx
          akx2=akx*akx
          read(1) (ci(i),i=1,nt)
          ci(ntp1)=ci(1)
          do 30 jf=1,ntn
            jfc=nt-jf+2
            f=(jf-1)*df
            dis=(f*f/v2) - akx2
            if(dis.gt.0.) go to 10
            co(jf)=0.
            co(jfc)=0.
            go to 30
10          akz=sqrt(dis)
            s=-f/v*akz
            r=akz/dkz + 1.
            it=int(r)
            itc=nt-it+2
            itc1=itc-1
            q=r-float(it)
c           interpolate
            co(jfc)=caveg(q,ci(it),ci(it+1))
            co(jf)=caveg(q,ci(itc),ci(itc1))
            co(jf)=co(jf)*s
            co(jfc)=co(jfc)*s
30        continue
40      write(2) (co(i), i=1,nt)
```

```
c
c       this do loop goes from kxnyqist+dkx to kxsample-dkx and
c       treats the second and third quadrants in figs. 3b.
c
        nxn1=nxn+1
        do 70 jx=nxn1,nx
          lx=nx-jx+1
          akx=lx*dkx
          akx2=akx*akx
          read(1) (ci(i),i=1,nt)
          ci(ntp1)=ci(1)
          do 60 jf=1,ntn
            jfc=nt-jf+2
            f=(jf-1)*df
            dis=(f*f/v2) - akx2
            if(dis.gt.0.) go to 50
            co(jf)=0.
            co(jfc)=0.
            go to 60
50          akz=sqrt(dis)
            s=-f/v*akz
            r=akz/dkz + 1.
            it=int(r)
            itc=nt-it+2
            itc1=itc-1
            q=r-float(it)
            co(jfc)=caveg(q,ci(it),ci(it+1))
            co(jf)=caveg(q,ci(itc),ci(itc1))
            co(jf)=co(jf)*s
            co(jfc)=co(jfc)*s
60        continue
70      write(2) (co(i),i=1,nt)
        write(6,106)
106     format(2x'fkdiffract finished'/)
        stop
        end
```