High Order Migration Operators for Laterally Homogeneous Media

Bert Jacobs and Francis Muir

Abstract

Implementation of a fourth order finite difference operator for downward continuation leads to a scheme which is economical, with roughly double the CPU costs of the 45-degree equation. The coefficients of the algorithm can be generated recursively.

Introduction

High dips are present in quantity on profiles so their migration is difficult. The 45-degree equation, a second-order scheme, may not be sufficient for this purpose. Both the third and fourth order schemes lead to pentadiagonal systems of linear equations. The fourth order scheme is therefore the next that should be considered when migrating in a medium that is laterally homogeneous.

The usual development of a finite difference scheme will not be followed here because the algebra needed to develop formulas for the coefficients is too tedious to get right. Instead a recursion for generating coefficients for equations of arbitrarily high order is developed.

Notation and Differentiation Operators

The operator notation used here will not be the standard one. It will be chosen so that symbols for the same type of operator in the discrete and continuous cases will be the same. Whenever possible, symbols will be chosen to reflect mathematical properties like causality and non-negative definiteness.

We begin with a causal differentiator with respect to time, D_t. A finite, causal, discrete differentiator has a diagonal made up of $1/\Delta t$'s and a sub-diagonal of $-1/\Delta t$'s, where Δt is the discretization interval size.

An anti-causal differentiator can also be defined. A finite, anti-causal, discrete differentiator has a diagonal full of $1/\Delta t$'s and a super-diagonal of $-1/\Delta t$'s. The symbol which will be used for this operator is D_{-t}.

A differentiator with respect to x can be similarly defined. Following Muir and Claerbout (this report), a rational form may be used instead. For example, a good first derivative approximation is given in terms of the x-axis delay operator Z by

$$D_x = \frac{1}{\Delta x} \frac{1 - Z}{(1 - \alpha) + \alpha Z}$$

Reasonable values of α lie between 1/2 and 1/10.9. A second derivative operator can be constructed from the first derivative

$$D_x^* D_x = \frac{1}{\Delta x^2} \frac{1 - Z - Z^* + |Z|^2}{1 - (\alpha - \alpha^*)(1 - Z)(1 - Z^*)}$$

We will not even use a first derivative with respect to x. This being so, we will introduce a new notation for the negative of the second derivative with respect to x, namely $|D_x|^2$. The negative of a second derivative is a symmetric, non-negative definite operator and this notation faithfully reflects these traits. In previous work, a discrete operator with -1,2,-1 on its sub-diagonal, diagonal, and super-diagonal, respectively, was denoted by T. An discrete approximation to the negative of a second derivative which uses T is

$$|D_x|^2 = \frac{1}{\Delta x^2} \frac{T}{I - \beta T} \tag{1}$$

where β is a positive, real number with a magnitude somewhere between 1/12 and 1/6.

Wave Equations

Migration is an implementation of a one-way wave-equation. To push upcoming waves down, the proper one-way wave-equation for the job is

$$D_{-z} P = -(\Lambda^2 D_{-t}^2 + |D_x|^2)^{1/2} P = -\left[\Lambda D_{-t} - \Lambda D_{-t} + (\Lambda^2 D_{-t}^2 + |D_x|^2)^{1/2}\right] P \tag{2}$$

The square root in (2) is defined to have a positive real part. The significance of this statement for the root an operator may not be clear yet. The root in (2) is operating on a normal operator which has an eigenfunction decomposition of the form $U^* diag(\lambda) U$ where U

is a unitary operator. The square root of this operator is now understood has an eigenfunction decomposition of the form $U^* diag(\sqrt{\lambda})\, U$ where each $\sqrt{\lambda}$ has a positive real part.

The minus signs in front of the square roots appear because the subject of equation (2) is an upgoing wave. The D_{-t}'s appear because (2) is intended for migrating rather than for modeling and migration is an anti-causal operator with respect to time. The development will be for laterally homogeneous velocity fields. This means that Λ, the acoustic slowness, will be a diagonal operator in x and that the placement of the Λ's in (2) with respect to the derivatives is not critical.

The numerical properties of our implementation of the one-way wave-equation will be considerably improved if we split equation (2) into two pieces.

1. $D_{-z} P \;=\; -\Lambda D_{-t} P \;=\; shift\ P$

2. $D_{-z} P \;=\; -\left[-\Lambda D_{-t} + (\Lambda^2 D_{-t}^2 + |\, D_x \,|^2)^{1/2} \right] P \;=\; undiffract\ P$

where the operator in the second line is an "undiffract" operator because we are migrating and migration is the inverse to diffraction. The basic recursion in a migration algorithm is the process of stepping a wavefield from a depth z to a depth $z + \Delta z$. When doing this, the first equation can be implemented approximately by using $P(x,z+\Delta z,t) \;=\; P(x,z,t - \Lambda \Delta z)$. The second equation in the split presents more of a problem. It has a formal solution of the form

$$P(x,z+\Delta z,t) \;=\; \exp\left\{ -\Delta z \left[-\Lambda D_{-t} + (\Lambda^2 D_{-t}^2 + |\, D_x \,|^2)^{1/2} \right] \right\} P(x,z,t) \qquad (3)$$

The Crank-Nicholson approximation of equation (3) is given by

$$P(x,z+\Delta z,t) \;=\; \frac{I - \left[(\Lambda^2 D_{-t}^2 + |\, D_x \,|^2)^{1/2} - \Lambda D_{-t} \right]}{I + \left[(\Lambda^2 D_{-t}^2 + |\, D_x \,|^2)^{1/2} - \Lambda D_{-t} \right]} P(x,z,t) \qquad (4)$$

The matrix divisions in equation (4) are justified because both the numerator and denominator are functions of $|\, D_x \,|^2$. Using the approximation of equation (1) and defining

$$A \;=\; \frac{\Delta z}{2} \Lambda D_{-t} \qquad n \;=\; \left(\frac{\Delta z}{2 \Delta x} \right)^2 T \qquad d \;=\; I - \beta T$$

changes equation (4) into the dimensionless form

$$P(x,z+\Delta z,t) \;=\; \frac{I - \left[\left(A^2 + \frac{n}{d} \right)^{1/2} - A \right]}{I + \left[\left(A^2 + \frac{n}{d} \right)^{1/2} - A \right]} P(x,z,t) \qquad (5)$$

The numerator and denominator of this rational matrix function need to be expanded as a continued fraction.

Denominator Approximants

The denominator of equation (5) can be represented as a continued fraction to get rid of the square root. Using the usual fraction and applying an equivalence transformation

$$I + \left[\left(A^2 + \frac{n}{d}\right)^{1/2} - A\right] = I + \cfrac{n}{2Ad + \cfrac{n}{2A + \cfrac{n}{2Ad + \cfrac{n}{2A + \cdots}}}} \qquad (6)$$

Equation (6) generates a series of approximants which satisfy the fundamental recurrence relations for continued fractions (see Appendix A or the first 16 pages of Wall's book on continued fractions). Denoting the jth approximant N_j^+ / D_j^+, the numerators N_j^+ satisfy

$$N_{-1}^+ = I \qquad N_0^+ = I \qquad N_1^+ = 2Ad + n$$

$$N_j^+ = 2AN_{j-1}^+ + nN_{j-2}^+ \qquad (j \text{ even}, j \geq 2)$$

$$N_j^+ = 2AdN_{j-1}^+ + nN_{j-2}^+ \qquad (j \text{ odd}, j \geq 3)$$

The denominators D_j^+ satisfy a second-order periodic recurrence as well. This recurrence is given by

$$D_{-1}^+ = 0 \qquad D_0^+ = I \qquad D_1^+ = 2Ad$$

$$D_j^+ = 2AD_{j-1}^+ + nD_{j-2}^+ \qquad (j \text{ even}, j \geq 2)$$

$$D_j^+ = 2AdD_{j-1}^+ + nD_{j-2}^+ \qquad (j \text{ odd}, j \geq 3)$$

Numerator Approximants

The numerator in the fraction in equation (5) can also be approximated by a continued fraction and its sequence of approximants. If the jth approximant is denoted N_j^- / D_j^- then it is found that the N_j^- and the D_j^- satisfy recurrences. In fact, N_j^- and D_j^- satisfy the same recurrences as N_j^+ and D_j^+, respectively. The only differences for the N's occur at the initialization steps. The first three N_j^-'s are

$$N_{-1}^- = I \qquad N_0^- = I \qquad N_1^- = 2Ad - n$$

The initializations for the sequence of D_j^-'s is precisely that of the D_j^+'s. Hence

$$D_j^+ = D_j^- \qquad (all\ j) \tag{7}$$

Recursions for High-Order One-Way Wave-Equations

Given the sequences of numerator and denominator approximants we can approximate equation (5) with yet another sequence of rational matrix functions. In terms of the four operators D_j^+, D_j^-, N_j^+, and N_j^-, the operator on the right-hand side of equation (5) can be written as

$$\frac{I - \left[\left(A^2 + \frac{n}{d}\right)^{1/2} - A\right]}{I + \left[\left(A^2 + \frac{n}{d}\right)^{1/2} - A\right]} = \frac{N_j^- / D_j^-}{N_j^+ / D_j^+} = \frac{N_j^-}{N_j^+} \tag{8}$$

where use has been made of equation (6).

Equation (8) implies that there exists a recursive way in which to generate the coefficients of migration schemes. If the jth approximation to the operator in equation (5) is $N_j / D_j = N_j^- / N_j^+$, then the operator will obey a set of recurrences.

$$B = \left(\frac{\Delta z}{2\Delta x}\right)^2 \qquad A = \frac{\Delta z}{2}\Lambda D_{-t}$$

$$N_{-1} = I \qquad N_0 = I \qquad N_1 = 2A(I - \beta T) + BT$$

$$N_j = 2AN_{j-1} + BTN_{j-2} \qquad (j\ even, j \geq 2)$$

$$N_j = 2A(I - \beta T)N_{j-1} + BTN_{j-2} \qquad (j\ odd, j \geq 3) \tag{9}$$

$$D_{-1} = I \qquad D_0 = I \qquad D_1 = 2A(I - \beta T) - BT$$

$$D_j = 2AD_{j-1} + BTD_{j-2} \qquad (j\ even, j \geq 2)$$

$$D_j = 2A(I - \beta T)D_{j-1} + BTD_{j-2} \qquad (j\ odd, j \geq 3)$$

These recurrences are useful in that they allow the programmer to build wave equations of successively higher order from the j=0 and j=1 terms. The j=1 terms give a 15-degree equation, the j=2 terms a 45-degree equation, the j=3 terms an equation of third order, and so on.

Second Derivatives for Wave Equations

One of the parameters which needs to be assigned a value is the β in equation (1). By varying β it is possible to vary the bandwidth and accuracy of the approximation. A choice of $\beta = 1/12$ makes the derivative as accurate as it can get in the vicinity of $k_x = 0$. If the logarithm of the error is to be minimaxed over wavelengths from DC to Nyquist then a value of 0.1326 should be used. In the program presented here, β is set equal to 1/12 for $\omega = 0$. β is set equal to 0.1326 for all frequencies $\omega > vk_N$, where k_N is the Nyquist frequency for the x-axis. Between 0 and vk_N, β is constrained to vary linearly from 1/12 to 0.1326.

Tests of this scheme yielded pleasing results. If the input is a zero-phase spike then the output remained zero phase. The "smile" of the output lacked the prominent fringes which a bad choice of β creates.

Wraparound Removal, DC, and Nyquist

Temporal DC and Nyquist frequency components cause trouble. Nyquist corresponds to neither positive nor negative continuous frequencies so it fouls up causality properties. DC is nettlesome because the wave equation does not propagate zero frequencies. Consider for instance the migration of a spike which is supposed to yield a semicircle. If the DC is passed, however, a prominent vertical streak is created as well. This streak can be eliminated by zeroing DC. The same is done to Nyquist.

The practice of zeroing DC and Nyquist is in contradiction with the usual method of roundoff removal. In theory, the seismogram at $t = 0$ is removed before diffracting at each z-step by subracting the average

$$\frac{1}{N}\left[Re\,(P(x,z,0)) + Re\,(P(x,z,\omega_{N/2})) + \sum_{j=1}^{N/2-1} 2Re\,(P(x,z,\omega_j)\right]$$

where N is the number of time points in the input and a power of two. Thus $\omega_j = 2\pi j/(N\,\Delta t)$. If DC and Nyquist are unavailable then the best average that is available to estimate the wavefield at $t = 0$ is

$$\frac{1}{N-2}\sum_{j=1}^{N/2-1} 2\mathrm{Re}(P(x,z,\omega_j)$$

Dip Filtering

A dip filter is obtained by altering the "foot" of the square root approximation used by a migration algorithm. We are more or less solving a partial differential equation with a dispersion relation

$$k_z = \omega\Lambda \cfrac{s^2}{2i + \cfrac{s^2}{2i + \cfrac{s^2}{2i + \cfrac{s^2}{2i}}}}$$

where $s = vk_x/\omega$. This does not have any dip filtering so we add this in next

$$k_z = \omega\Lambda \cfrac{s^2}{2i + \cfrac{s^2}{2i + \cfrac{s^2}{2i + \cfrac{s^2}{2i(\psi + i\varepsilon)}}}} \tag{10}$$

This is at variance with most previous SEP work in that it makes the dip filtering parameter ε frequency dependent. The reason for this is that the dip attenuation simplifies with this choice. The real part of k_z from equation (10) is

$$\omega\Lambda \frac{-\varepsilon s^8}{(s^4 - (4 + 8\psi)s^2 + 16\psi)^2 + 64\varepsilon^2(2 - s^2)^2} \tag{11}$$

which is a function of s alone (if the multiplicative factor $\omega\Lambda$ is ignored). Since s is the sine of the propagation angle equation (11) is an unusually simple form for the response of a migration dip filter.

Migration Examples

The fourth order operator was used to generate a migration impulse response. The migration and plotting parameters were as follows:

$dx = 20.0 \quad dz = 20.0 \quad dt = 0.004$

$nx = 129 \quad nz = 128 \quad nt = 64$

$\varepsilon = 0.0625 \quad \psi = 0.4375 \quad v = 10000. \quad clip = 2.0$

A seismogram was placed on the 65th trace, centered at time point 32. The waveform was a three point wavelet of the form -1,2,-1 to cut out zero frequency. The migration output is in Figure 1.

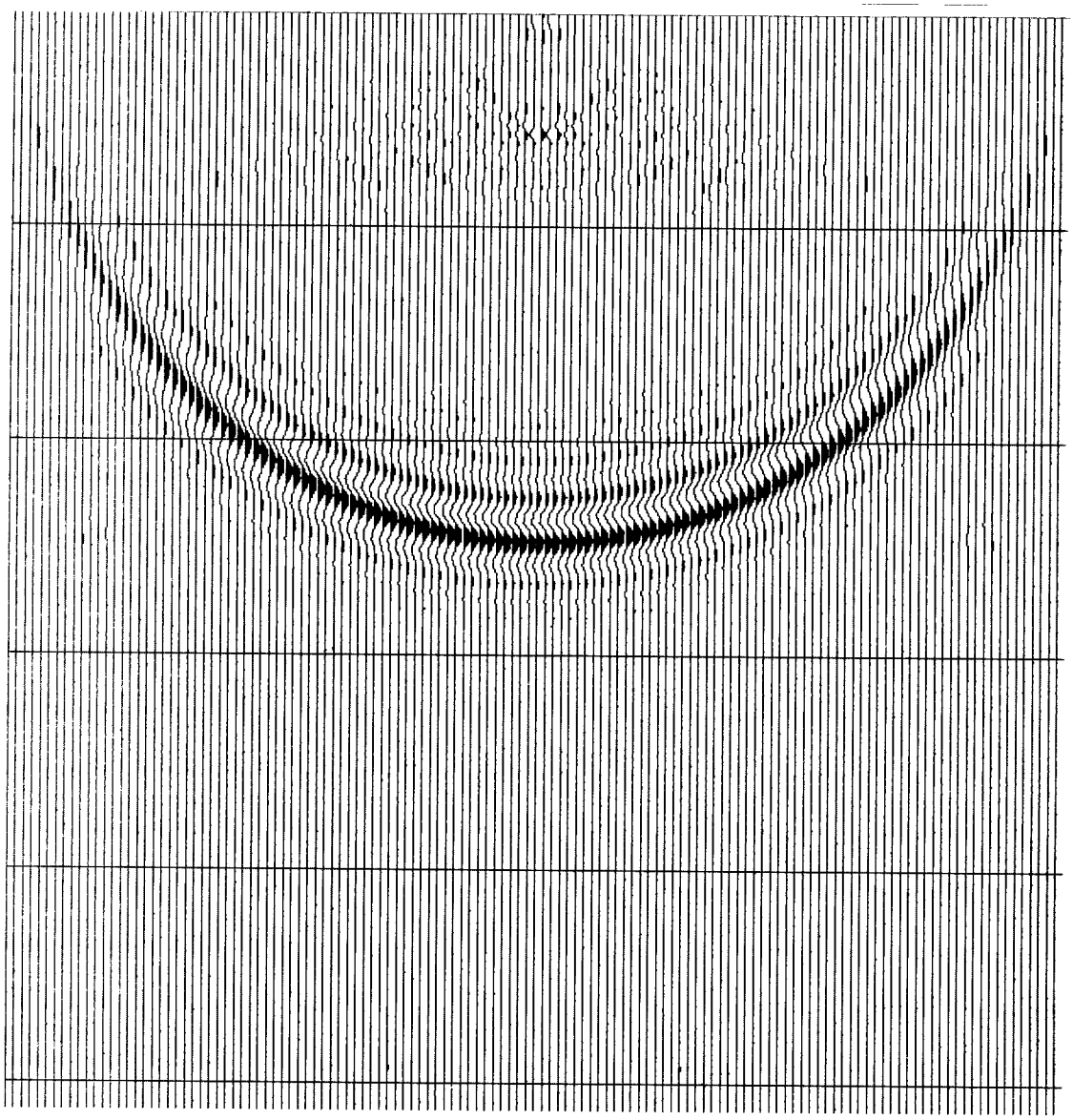

FIG. 1. This impulse response was generated by a fourth-order migration algorithm. The coefficients ε and ψ were chosen to get the dispersion curve to match at 90 degree dips.

An equivalent migration using the 45-degree equation was done with the same parameters, with the exception of ε and ψ which were given by

$$\varepsilon = 0.125 \qquad \psi = 0.375$$

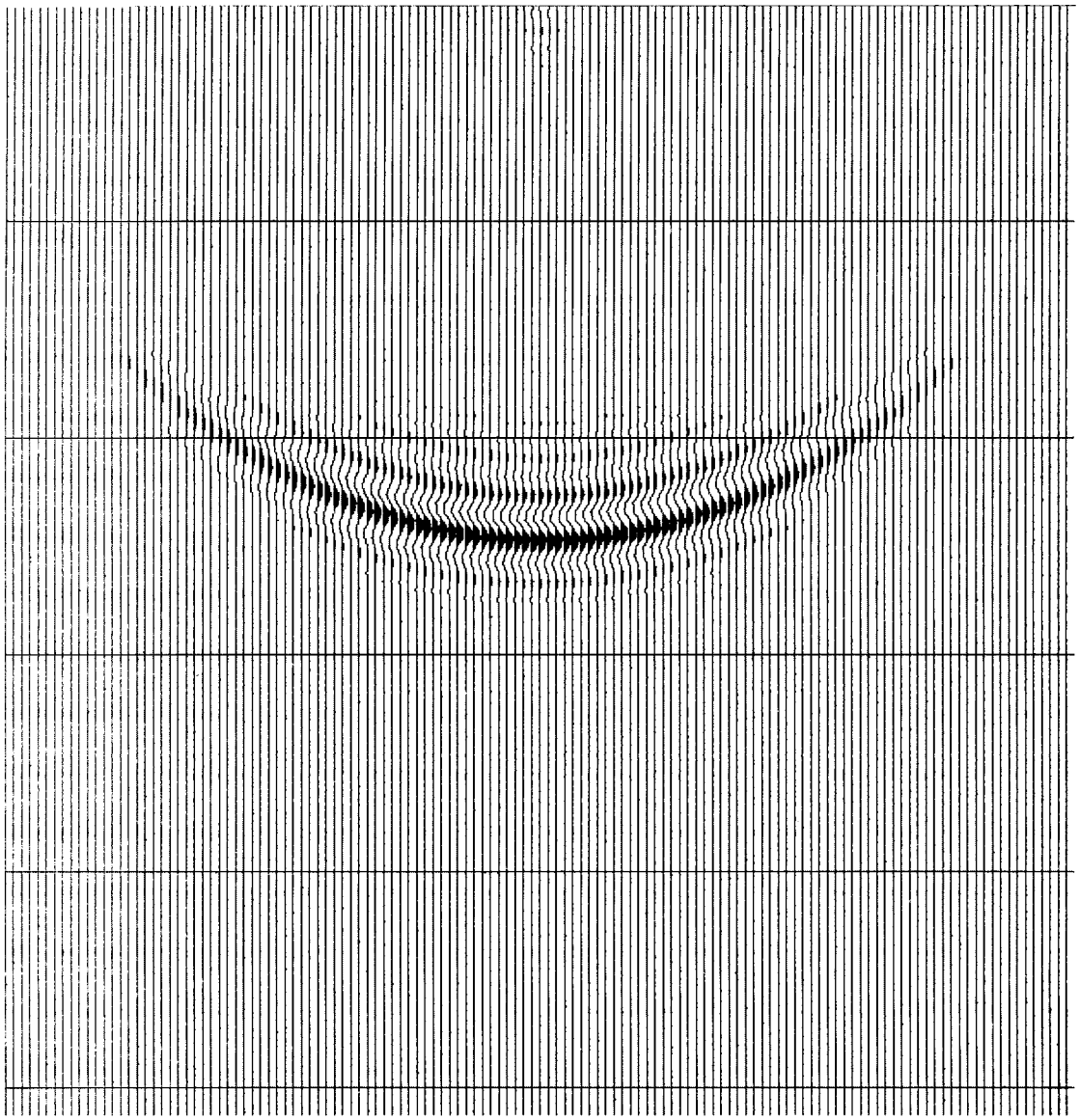

FIG. 2. A 45-degree migration impulse response has poor behavior at high dips.

Appendix A

Given a continued fraction we can usually form an infinite sequence of approximants by truncating the fraction. Borrowing heavily from Wall, suppose we are given a sequence of transformations, $t_0, t_1, t_2, \cdots$ defined by

$$t_0(w) = b_0 + w \qquad t_p(w) = \frac{a_p}{b_p + w}$$

where the a_p's and b_p's are complex numbers and w is a complex variable. a_p and b_p are

called the pth partial numerator and partial denominator, respectively. The continued fraction that corresponds to this series of transformations is

$$\lim_{n\to\infty} t_0 t_1 \cdots t_n(0) = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \cdots}}}$$

The nth approximants of this continued fraction are given in terms of nth numerator A_n, and the nth denominator B_n by

$$t_0 t_1 \cdots t_n(0) = \frac{A_n}{B_n}$$

so that the first few approximants are

$$b_0 \qquad b_0 + \frac{a_1}{b_1} \qquad b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2}}$$

The numerators and denominators of the approximants satisfy the fundamental recurrence formulas

$$A_{-1} = 1 \qquad B_{-1} = 0 \qquad A_0 = b_0 \qquad B_0 = 1$$

$$A_{p+1} = b_{p+1}A_p + a_{p+1}A_{p-1} \qquad (p = 0,1,2,\cdots)$$

$$B_{p+1} = b_{p+1}B_p + a_{p+1}B_{p-1} \qquad (p = 0,1,2,\cdots)$$

These formulas can be proved by induction.

REFERENCES

Godfrey, B. and Jacobs, B. (1979), A Program for Stable Migration, SEP-16, pp. 109-19.

Morley, Larry (1979), Numerical Viscosity Considerations for the Monochromatic 45-Degree Equation, SEP-16, pp. 109-19.

Wall, H.S. (1948), Analytic Theory of Continued Fractions: Bronx, N.Y., Chelsea Publishing Co.

```
#    Migration (z-variable velocity) program using
#    recurrences to find coefficients.  The foot of
#    continued fraction for the square root is doctored.
#    Fourier transform conventions:
#     F.T.      : P(x,w,z) = sum[P(x,t,z)exp(-iwt)]
#     F.T.**-1  : P(x,w,z) = (nt**-1) sum[P(x,t,z)exp(iwt)]
#    nx = no. x-grid pts          nt = no. t-grid pts
#    nz = no. dz steps            nw = nt/2+1 = no. frequencies
#    dx = grid interval in x      dt = grid interval in t
#    dz = z-step size             dw = frequency interval
#    eps = dip filtering > 0      psi = phase matching
#    beta = 2nd x-derivative coefficient, bmin < beta < bmax
#    pf: P(x,w,z=0)  - ifc is file address       : input
#    sc: P(x,w,z>0)  - ifs is file address       : scratch
#    mg: P(x,t=0,z)  - ifm is file address       : output
#    nx4 = bytes in real vector of nx elements
#    nx8 = bytes in complex vector of nx elements
#    wsc = scaling parameter for wraparound removal
#
common /global/msc,mx,my,q,r,s
complex*8 msc(3,5),mx(3,5),my(3,5),q(256),r(256),s(256)
complex*8 ml(3,5),mr(3,5),p(256),rhs(256),t(256)
complex*8 tc(256)
real*4 tr(256),vr
real*4 beta,bmax,bmin,dt,dx,dw,dz,eps,pi,rmig(256),psi
real*4 sl,v,w,wc,wrap(256),wsc
integer ichk,ifc,ifm,ifs,ifv,ir,il,iw,iz,nor,nread
integer nseek,nt,nw,nwrite,nx,nx4,nx8,nz,ix
integer cclose,cread,cwrite,cseek
#
#
call begin(ifc,ifs,ifm,ifv,nx,nt,nz,dx,dt,dz,eps,psi,nor)
bmax = 0.1326
bmin = 1.0/12.0
nw = nt/2 + 1
pi = 3.14159265
dw = 2.*pi/(nt*dt)
nx4 = 4*nx
nx8 = 8*nx
ir = 1
il = -1
wsc = 1.0/(nt-2.)
#    Copy the input and initialize wrap-around
#    Remove DC and Nyquist w's
do iw = 1,nw {
        nread = cread(ifc,tc,nx8)
        do ix = 1,nx
                t(ix) = tc(ix)
        if (iw == 1)
                call zero(2*nx,t)
        if (iw == nw)
                call zero(2*nx,t)
        call zapt0(rmig,t,iw,nw,nx)
        nwrite = cwrite(ifs,t,nx8)
        }
ichk = cclose(ifc)
```

```
call rscal(rmig,wrap,nx,wsc)
#
#   Downward continue in z
#   Read in the velocity, zero migration accumulator
#   Find wc, the w where x-aliasing occurs
do  iz = 1,nz {
        nread = cread(ifv,vr,4)
        v = vr
        sl = 1.0/v
        call zero(nx,rmig)
        wc = v*pi/dx
#   Loop over frequencies, treat DC differently
        do iw = 1,nw {
                nseek = cseek(ifs,(iw-1)*nx8,0)
                nread = cread(ifs,p,nx8)
                if (iw == 1)
                        call zero(2*nx,t)
                else if (iw == nw)
                        call zero(2*nx,t)
                else {
                        w = (iw-1.)*dw
                        if (w < wc)
                                beta = bmin+w*(bmax-bmin)/wc
                        else
                                beta = bmax
#   Remove t=0 component from p, implement phase shift,
#   then diffract
                        call unwrap(p,wrap,nx)
                        call cshift(sl,p,w,dz,nx)
                call mtrx(mr,ir,sl,w,eps,dx,dz,beta,psi,nor)
                call rhsv(mr,p,rhs,nx)
                call mtrx(ml,il,sl,w,eps,dx,dz,beta,psi,nor)
                call pentc(nx,t,ml,rhs)
                        }
#   Sum over w's to get t=0, save all w's in scratch file
                call zapt0(rmig,t,iw,nw,nx)
                nseek = cseek(ifs,(iw-1)*nx8,0)
                nwrite = cwrite(ifs,t,nx8)
                }
#   Save migrated output and compute wrap-around
        do ix = 1,nx
                tr(ix) = rmig(ix)
        nwrite = cwrite(ifm,tr,nx4)
        call rscal(rmig,wrap,nx,wsc)
        }
#   Close all files
ichk = cclose(ifm)
ichk = cclose(ifs)
ichk = cclose(ifv)
stop
end


#  Multiplies a real vector by a real scalar:
#  y(i)=x(i)*sc for i=1,2,...,n
subroutine rscal(x,y,n,sc)
```

```
real*4 sc,x(1),y(1)
integer i,n
do i = 1,n
        y(i) = sc*x(i)
return
end



#  Fills a real vector nx long with zeros
subroutine zero(nx,x)
real*4 x(1)
integer ix,nx
do ix = 1,nx
        x(ix) = 0.0
return
end



#   Subtract wrap-around from p
subroutine unwrap(p,wrap,nx)
integer nx
complex*8 p(1)
real*4 wrap(1)
integer ix
do ix = 1,nx
        p(ix) = p(ix) - wrap(ix)
return
end


#   Phase shift implementation
#   P <-- P exp( w*dz/v )
subroutine cshift(sl,p,w,dz,nx)
integer nx
real*4 dz,sl,w
complex*8 p(1)
integer ix
complex*8 csh
complex*8 cmplx
csh = exp(cmplx(0.0,w*dz*sl))
do ix = 1,nx
        p(ix) = p(ix)*csh
return
end


#   Sum frequencies to get t=0
#   rmig = int from w=0 to Nyquist of Re(t(w))
subroutine zapt0(rmig,t,iw,nw,nx)
real*4 rmig(1)
complex*8 t(1)
integer iw,nw,nx
real*4 fact
integer ix
```

```
if (iw == 1)
        fact = 0.0
else if (iw == nw)
        fact = 0.0
else
        fact = 2.0
do ix = 1,nx
        rmig(ix) = rmig(ix) + fact*real(t(ix))
return
end


#   Sets up propagation matrices
#   ior = order of recurrence
#   a and b are recursion parameters
#   al is the parameter 'a' modified for dip filtering
#   c is used in second derivative approximation
#   Odd & even recursion steps have different coefficients
subroutine mtrx(m,isin,sl,w,eps,dx,dz,beta,psi,nor)
common /global/msc,mx,my,q,r,s
complex*8 msc(3,5),mx(3,5),my(3,5),q(256),r(256),s(256)
complex*8 a,al,c,eye,m(3,5),cmplx
real*4 b,bs,sl,w,eps,dx,dz,beta,psi
integer iodd,ior,isin,nor
eye = cmplx(0.0,1.0)
a = eye*w*dz*sl
al = a*(psi + eye*eps)
b = (0.50*dz/dx)**2
bs = isin*b
c = -a*beta
call iden(mx)                       #  ior = -1
call iden(m)                        #  ior = 0
ior = 1                     #  ior = 1
if (ior == nor)
        a = al
call recuro(m,a,c,bs)
iodd = 0
if (nor > 1) {                      #  ior > 1
        do ior = 2,nor {
                if (ior == nor)  # dip filter the foot
                        a = al
                if (iodd == 0) {   # even recursion
                        call recure(m,a,b)
                        iodd = 1
                        }
                else {                  # odd recursion
                        call recuro(m,a,c,b)
                        iodd = 0
                        }
                }
        }
return
end


#   Recurrence for coefficients
```

```
#   M is the current set, Mx is the last set, My is the
#   next set.  My = x1*M + x2*T*M + x3*T*Mx
#   After calculations M --> Mx , My --> M
#   and My is ready for more scratch pad work.
subroutine recuro(m,x1,x2,x3)
common /global/msc,mx,my,q,r,s
complex*8 msc(3,5),mx(3,5),my(3,5),q(256),r(256),s(256)
complex*8 m(3,5),x1,x2
real*4 x3
call tmul(mx,msc)                  # dump T*Mx in Msc
call rmul(msc,x3,msc)              # dump x3*T*Mx in Msc
call tmul(m,my)                    # dump T*M in My
call cmul(my,x2,my)                # dump x2*T*M in My
call addr(msc,my,msc)              # dump x2*T*M+x3*T*Mx in Msc
call cmul(m,x1,my)                 # dump x1*M in My
call movr(m,mx)                    # move M --> Mx
call addr(msc,my,m)                # move x1*M+x2*T*M+x3*T*Mx
return                             #                --> M
end


#   Recurrence for coefficients
#   M is the current set, Mx is the last set, My is the
#   next set.  My = x1*M + x3*Mx
#   After calculations M --> Mx, My --> M
#   and My is ready for more scratch pad work.
subroutine recure(m,x1,x3)
common /global/msc,mx,my,q,r,s
complex*8 msc(3,5),mx(3,5),my(3,5),q(256),r(256),s(256)
complex*8 m(3,5),x1
real*4 x3
call tmul(mx,msc)                  # dump T*Mx in Msc
call rmul(msc,x3,msc)              # dump x3*T*Mx in Msc
call cmul(m,x1,my)                 # dump x1*M in My
call movr(m,mx)                    # move M --> Mx
call addr(msc,my,m)                # dump x1*M+x3*T*Mx in M
return
end


#
subroutine addr(m1,m2,m3)
complex*8 m1(3,5),m2(3,5),m3(3,5)
integer ic,ir
do ir = 1,3 {
        do ic = 1,5 {
                m3(ir,ic) = m1(ir,ic) + m2(ir,ic)
                }
        }
return
end


#
subroutine movr(min,mout)
complex*8 min(3,5),mout(3,5)
```

```
integer ic,ir
do ic = 1,5 {
        do ir = 1,3 {
                mout(ir,ic) = min(ir,ic)
                }
        }
return
end


#
subroutine cmul(min,sc,mout)
complex*8 min(3,5),mout(3,5),sc
integer ic
do ic = 1,3
        mout(1,ic) = sc*min(1,ic)
do ic = 1,4
        mout(2,ic) = sc*min(2,ic)
do ic = 1,5
        mout(3,ic) = sc*min(3,ic)
return
end


subroutine rmul(min,sc,mout)
complex*8 min(3,5),mout(3,5)
real*4 sc
integer ic
do ic = 1,3
        mout(1,ic) = sc*min(1,ic)
do ic = 1,4
        mout(2,ic) = sc*min(2,ic)
do ic = 1,5
        mout(3,ic) = sc*min(3,ic)
return
end


#   Multiply the input coefficient matrix by T, the
#   negative of a second-difference matrix
subroutine tmul(min,mout)
complex*8 min(3,5),mout(3,5)
integer ic
do ic = 1,3
        mout(1,ic) = min(1,ic)-min(2,ic)
do ic = 1,4
        mout(2,ic) = 2.0*min(2,ic)-min(1,ic)-min(3,ic)
mout(3,1) = 2.0*min(3,1) - min(2,1)
do ic = 2,5
        mout(3,ic) = 2.0*min(3,ic)-min(2,ic)-min(3,ic-1)
return
end



#   Initialize M (3 by 5 complex*8) to I
```

```
subroutine iden(m)
complex*8 m(3,5)
integer i,j
do i = 1,3 {
        do j = 1,5 {
                m(i,j) = 0.0
                }
        }
m(1,1) = 1.0
m(2,2) = 1.0
m(3,3) = 1.0
return
end


#    Calculate the rhs vector
subroutine rhsv(mr,p,rhs,nx)
complex*8 mr(3,5),p(1),rhs(1)
integer i,nx
rhs(1) = mr(1,1)*p(1)+mr(1,2)*p(2)+mr(1,3)*p(3)
rhs(2) = mr(2,1)*p(1)+mr(2,2)*p(2)+mr(2,3)*p(3)+mr(2,4)*p(4)
do i = 3,nx-2 {
        rhs(i) = mr(3,1)*p(i-2)+mr(3,2)*p(i-1)+mr(3,3)*p(i)
        rhs(i) = rhs(i)+mr(3,4)*p(i+1)+mr(3,5)*p(i+2)
        }
rhs(nx-1) = mr(2,4)*p(nx-3)+mr(2,3)*p(nx-2)+mr(2,2)*p(nx-1)
rhs(nx-1) = rhs(nx-1)+mr(2,1)*p(nx)
rhs(nx) = mr(1,3)*p(nx-2)+mr(1,2)*p(nx-1)+mr(1,1)*p(nx)
return
end



#   Open or create files named on  terminal
#   Grab input parameters from the terminal
subroutine begin(ifc,ifs,ifm,ifv,nx,nt,nz,dx,dt,dz,eps,psi,nor)
integer ifc,ifm,ifs,ifv,iperm,nor,nt,nx,nz
real*4 dt,dx,dz,eps,psi
real*4 far
integer iarl
iperm = 6*64+6*8+4
write(6,07)
write(6,08)
call openr(1,0,ifc)
call creatr(2,iperm,2,ifs)
call creatr(3,iperm,1,ifm)
call openr(4,0,ifv)
nx = iarl(5)
nt = iarl(6)
nz = iarl(7)
dx = far(8)
dt = far(9)
dz = far(10)
eps = far(11)
psi = far(12)
nor = iarl(13)
```

```
write(6,09) nx,nt,nz
write(6,10) dx,dt,dz
write(6,12) eps,psi
write(6,13) nor
07     format(1x)
08     format(3x,"RECURSIVE COEFFICIENT MIGRATION")
09     format(3x,"Nx=",i10,1x,"Nt=",i10,1x,"Nz=",i10)
10     format(3x,"Dx=",f10.4,1x,"Dt=",f10.4,1x,"Dz=",f10.4)
12     format(3x,"Eps=",f10.4,1x,"Psi=",f10.4)
13     format(3x,"Order=",i4)
return
end


#    Adapted from Walt Lynn's algorithm in SEP-15
#    Pentc solves a constant coefficient pentadiagonal system :
#     a*t(k-2)+b*t(k-1)+c*t(k)+d*t(k+1)+e*t(k+2) = rhs(k)
#    for k=1,2,...,n
#    The first two and last two rows are allowed anomolous
#    coefficients
#    Uses the recursion relation:
#     t(k) = q(k)*t(k+2)+r(k)*t(k+1)+s(k)*t(k)
subroutine pentc(n,t,ml,rhs)
common /global/msc,mx,my,q,r,s
complex*8 msc(3,5),mx(3,5),my(3,5),q(256),r(256),s(256)
complex*8 rhs(1),t(1),ml(3,5)
integer n
complex*8 a,b,c,d,e,c1,d1,e1,b2,c2,d2,e2,an1,bn1,cn1,dn1
complex*8 an,bn,cn,ct,den,g1,g2,g3,h1,h2,h3
integer i,i1,i2,n1,n2,n3
c1 = ml(1,1)
d1 = ml(1,2)
e1 = ml(1,3)
b2 = ml(2,1)
c2 = ml(2,2)
d2 = ml(2,3)
e2 = ml(2,4)
a = ml(3,1)
b = ml(3,2)
c = ml(3,3)
d = ml(3,4)
e = ml(3,5)
an1 = ml(2,4)
bn1 = ml(2,3)
cn1 = ml(2,2)
dn1 = ml(2,1)
an = ml(1,3)
bn = ml(1,2)
cn = ml(1,1)
#
n1 = n - 1
n2 = n - 2
n3 = n - 3
q(1) = -e1/c1
r(1) = -d1/c1
s(1) = rhs(1)/c1
```

```
den = 1.0/(c2 + b2*r(1))
q(2) = -e2*den
r(2) = -(b2*q(1) + d2)*den
s(2) = (rhs(2) - b2*s(1))*den
do i = 3,n2 {
        i1 = i - 1
        i2 = i - 2
        ct = a*r(i2) + b
        den = 1.0/(ct*r(i1) + a*q(i2) + c)
        q(i) = -e*den
        r(i) = -(q(i1)*ct + d)*den
        s(i) = (rhs(i) - s(i1)*ct - a*s(i2))*den
        }
g1 = an*q(n2) + cn
g2 = an*r(n2) + bn
g3 = rhs(n) - an*s(n2)
ct = an1*r(n3) + bn1
h1 = ct*q(n2) + dn1
h2 = an1*q(n3) + cn1 + ct*r(n2)
h3 = rhs(n1) - an1*s(n3) - ct*s(n2)
den = 1.0/(g1*h2 - g2*h1)
t(n) = (g3*h2 - g2*h3)*den
t(n1) = (g1*h3 - g3*h1)*den
do i = n2,1,-1 {
        i1 = i + 1
        i2 = i + 2
        t(i) = q(i)*t(i2) + r(i)*t(i1) + s(i)
        }
return
end
```

There was a young poet named Dan,
Whose poetry never would scan.
When told this was so,
He said,"yes, I know,
It's because I try to put every possible syllable into that last line that I can."

Niklaus Wirth has lamented that, whereas Europeans pronounce his name correctly (Ni-klows Virt), Americans invariably mangle it into (Nick-les Worth). Which is to say that Europeans call him by name, but Americans call him by value.

WHERE CAN THE MATTER BE

Oh, dear, where can the matter be
When it's converted to energy?
There is a slight loss of parity.
Johnny's so long at the fair.

A truly wise man never plays leapfrog with a unicorn.

Kleptomaniac: A rich thief.

On his first day as a bus driver, Maxey Eckstein handed in receipts of $65. The next day his take was $67. The third day's income was $62. But on the fourth day, Eckstein emptied no less than $283 on the desk before the cashier.

"Eckstein!" exclaimed the cashier. "This is fantastic. That route never brought in money like this! What happened?"

"Well, after three days on that cockamany route, I figured business would never improve, so I drove over to Fourteenth Street and worked there. I tell you, that street is a gold mine!"

Monday: In Christian countries, the day after the baseball game.

You may be recognized soon. Hide.

"The shortest distance between two points is under construction."
-- Noelie Altito

Do not sleep in a eucalyptus tree tonight.

A penny saved is ridiculous.