Stanford Exploration Project, Report 73, November 18, 1997, pages 385–391

Introduction to Kirchhoff migration programs

Jon F. Claerbout[1]

keywords: *modeling, algorithm, Fourier transform*

ABSTRACT

In the simplest zero-offset Kirchhoff program (which is very slow) loops may be reordered arbitrarily. I reorder the code to avoid computing results that lie off the bottom of mesh and to reuse square roots thus making the code run much faster. An example shows that diffraction followed by migration gives low frequency amplification.

Neither IEI (Claerbout, 1985) nor PVI (1992) contains runnable, conjugate Kirchhoff code. This article fills the gap. Eventually this material will appear in a revised IEI. Meanwhile, I plan to put to use the fast version below in iterative migrations for parameter and missing data estimation.

TUTORIAL KIRCHHOFF CODE

Subroutine `kirchslow()` below is the best tutorial Kirchhoff migration-modeling program I could devise. A nice feature of this program is that it works OK while the edge complications do not clutter it. The program copies information from data space `data(it,iy)` to model space `modl(iz,ix)` or vise versa. Notice that of these four axes, three are independent (stated by loops) and the fourth is derived by the circle-hyperbola relation $t^2 = \tau^2 + x^2/v^2$. Migration programs are said to be "hyperbola-like" if τ is given and t is solved for, and "semicircle-like" if t is given and τ is solved for. Subroutine `kirchslow()` is hyperbola-like and for `conj=0`, information is copied from model space to data space, i.e. from the hyperbola top to its flanks. Otherwise data flows from the flanks to the top.

```
# Kirchhoff migration and diffraction.  (tutorial, slow)
#
subroutine kirchslow(   conj, add,  velocity, t0,dt,dx, nt,nx, modl, data)
integer ix,iy,it,iz,nz, conj, add,                      nt,nx
real x0,y0,dy,z0,dz,t,x,y,z,hs,     velocity, t0,dt,dx, modl(nt,nx), data(nt,nx)
call conjnull(          conj, add,                      modl,nt*nx,  data,nt*nx)
```

[1]**email:** not available

```
x0=0.;  y0=0;  dy=dx;  z0=t0;  dz=dt; nz=nt
do ix= 1, nx {  x = x0 + dx * (ix-1)
do iy= 1, nx {  y = y0 + dy * (iy-1)
do iz= 1, nz {  z = z0 + dz * (iz-1)            # z = travel-time depth
        hs=      (x-y) / velocity
        t = sqrt( z * z  +  hs * hs )
        it = 1.5 + (t-t0) / dt
        if( it <= nt )
                if( conj == 0 )
                        data(it,iy) = data(it,iy) + modl(iz,ix)
                else
                        modl(iz,ix) = modl(iz,ix) + data(it,iy)

        }}}
return; end
```

The three loops in subroutine `kirchslow()` may be interchanged at will without changing the result. To emphasize this flexibility, the loops are set at the same indentation level. We tend to think of fixed values of the outer two loops and then describe what happens on the inner loop. For example, if the outer two loops are those of the model space `modl(iz,ix)`, then for `conj=1` the program sums data along the hyperbola into the "fixed" point of model space. When loops are reordered, we think differently and opportunities arise for speed improvements.

FAST KIRCHHOFF CODE

Subroutine `kirchslow()` is very slow in the normal case where $x_{\max} \gg vt_{\max}$ because then many values of t are computed beyond $t_{\max}$. To avoid this, we notice that for fixed offset (`ix-iy`) and variable depth `iz`, as depth increases, time `it` eventually goes beyond the bottom of the mesh and, as soon as this happens, it will continue to happen for all larger values of `iz`. Thus we can `break` out of the `iz` loop the first time we go off the mesh to avoid computing anything beyond as shown in subroutine `kirchfast()`. (Some quality compromises, limiting the aperature or the dip, also yield speedup, but we avoid those.) Another big speedup arises from reusing square roots. Since the square root depends only on offset and depth, once computed it can be used for all `ix`. Finally, these changes of variables have left us with more complicated side boundaries, but once we work these out, the inner loops can be devoid of tests and in `kirchfast()` they are in a form that is highly optimizable by many compilers.

```
# Kirchhoff migration and diffraction.  (greased lightning)
#
subroutine kirchfast(   conj, add,  velocity, t0,dt,dx, nt,nx, modl, data)
integer ix,iz,it,ih,    conj, add,                       nt,nx,     xstart, xend
real    amp,t,z,h,                  velocity, t0,dt,dx,modl(nt,nx), data(nt,nx)
call conjnull(          conj, add,                      modl,nt*nx,  data,nt*nx)
do ih= -nx, nx {        h = dx * ih                      # h = offset
```

```
       do iz= 2, nt {  z = t0 + dt * (iz-1)            # z = travel-time depth
                       t = sqrt( z**2  +  (h/velocity)**2 )
                       it = 1.5 + (t - t0) / dt
            if( it > nt )  break
               amp = (z / t)  *  sqrt( nt*dt / t )
               xstart =  1-ih;         xstart = max0(  1, xstart)
               xend   = nx-ih;         xend   = min0( nx, xend)
               if( conj == 0 )
                       do ix= xstart, xend
                               data(it,ix+ih)=data(it,ix+ih)+modl(iz,ix   )*amp
               else
                       do ix= xstart, xend
                               modl(iz,ix   )=modl(iz,ix   )+data(it,ix+ih)*amp
               }
       }
return; end
```

Originally the two Kirchhoff programs produced identical output, but finally I could not resist adding an important feature to the fast program, scale factors $z/t = \cos\theta$ and $1/\sqrt{t}$ that are described elsewhere. The fast program, with a few changes, allows for velocity variation with depth. (The `break` should be conditioned on the maximum velocity followed by the `if()` conditioned on the velocity at the current depth.) When velocity varies laterally we need to go back to the beginning.

Figure 1 shows an example. The model includes dipping beds, syncline, anticline, fault, unconformity, and buried focus. The result is as expected with a "bow tie" at the buried focus. On a video screen, I can see hyperbolic events originating from the unconformity and the fault. At the right edge are a few faint edge artifacts. We could have reduced or eliminated these edge artifacts if we had extended the model to the sides with some empty space.

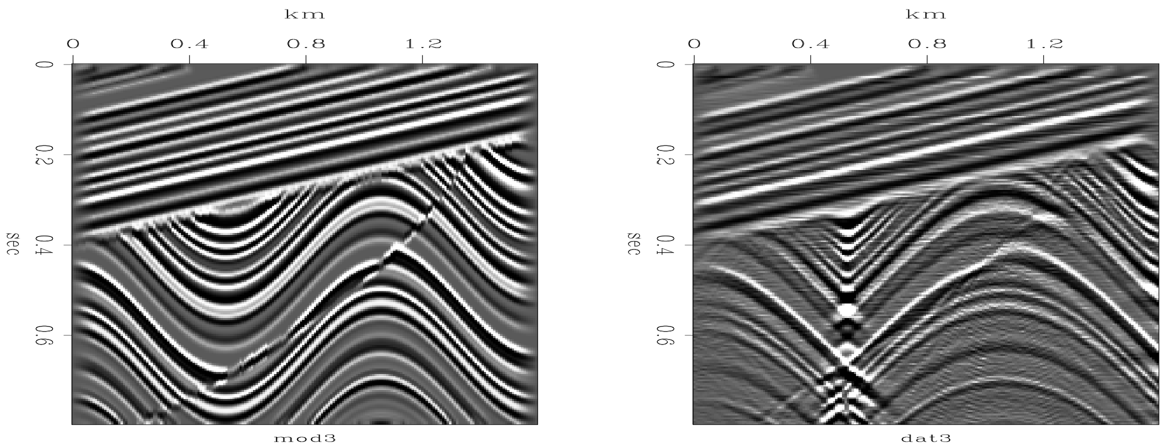

Figure 1: Left is the model. Right is diffraction to synthetic data. jon1.kirch-kfgood [ER]

ARTIFACTS

Reconstructing the earth model with the conjugate option in `kirchfast()` on page 386 yields the result in Figure 2. The reconstruction generally succeeds but is imperfect

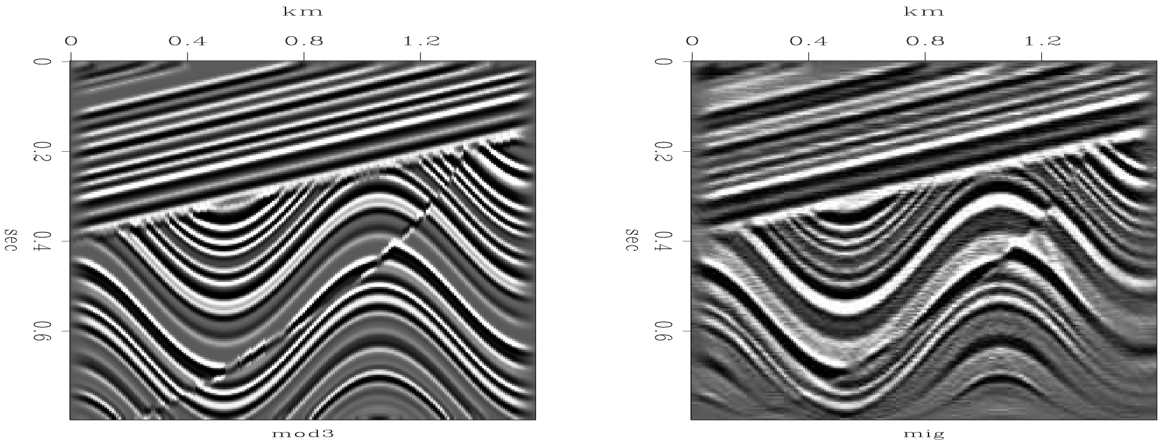

Figure 2: Left is the original model Right is the reconstruction. jon1.kirch-skmig [ER]

in a number of interesting ways. Near the bottom and right side, the reconstruction fades away especially where the dips are steeper. Bottom fading results because in modeling the data we abandoned arrivals after a certain maximum time. Thus energy needed to reconstruct dipping beds near the bottom was abandoned. Likewise along the side we abandoned rays shooting off the frame.

Difficult migrations are well known for producing semicircular reflectors. Here we have controlled everything fairly well so none are obvious, but on a video screen I see some semicircles.

Next is the problem of the spectrum. Notice in Figure 2 that the reconstruction lacks the sharp crispness of the original. It can be shown mathematically that the spectrum of our reconstruction loses high frequencies by a scale of about $1/|\omega|$. Philosophically, we can think of the integration as boosting low frequencies. Figure 3 shows the average over x of the relevant spectra. First, notice the high frequencies are weak because there is little high frequency energy in the original model. Then notice that our cavalier approach to interpolation created more high frequency energy. Finally, notice that multiplying the spectrum of our migrated model by frequency, f, brought the important part of the spectral bands into agreement. This suggest applying an $|\omega|$ filter to our reconstruction, or $\sqrt{-i\omega}$ operator to both the modeling and the reconstruction. We will take that up somewhere else.

Neither of these Kirchhoff codes addresses the issue of spatial aliasing. Spatial aliasing is a vexing issue of numerical analysis. The Kirchhoff codes shown here do not work as expected unless the space mesh size is suitably more refined than the time mesh. Figure 4 shows an example of forward modeling with a spatial mesh of 50

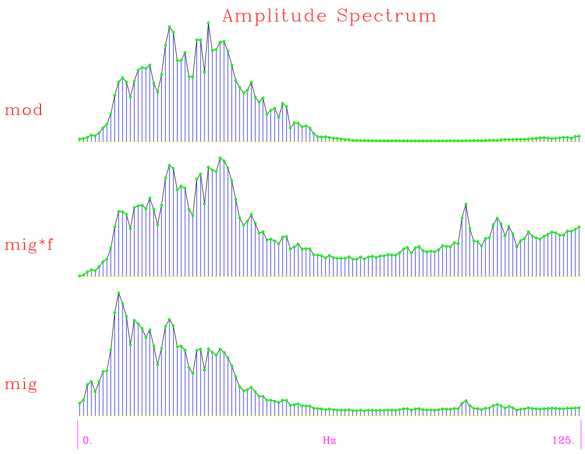

Figure 3: Top is the spectrum of the left side of Figure 2. Bottom is the spectrum of the right side of Figure 2. Middle is bottom times frequency. jon1.kirch-kirspec [ER]

and 100 points. (Previous figures used 200 points on space. All use 200 mesh points on the time.) Subroutine `kirchfast()` on page 386 does interpolation by moving values to the nearest neighbor of the theoretical location. Had we taken the trouble to interpolate the two nearest points, our results would have been a little better, but the basic problem (resolved in jon/res/trimo) would remain.

REFERENCES

Claerbout, J. F., 1985, Imaging the Earth's Interior: Blackwell Scientific Publications.

Claerbout, J. F., 1992, Earth Soundings Analysis: Processing Versus Inversion: Blackwell Scientific Publications.

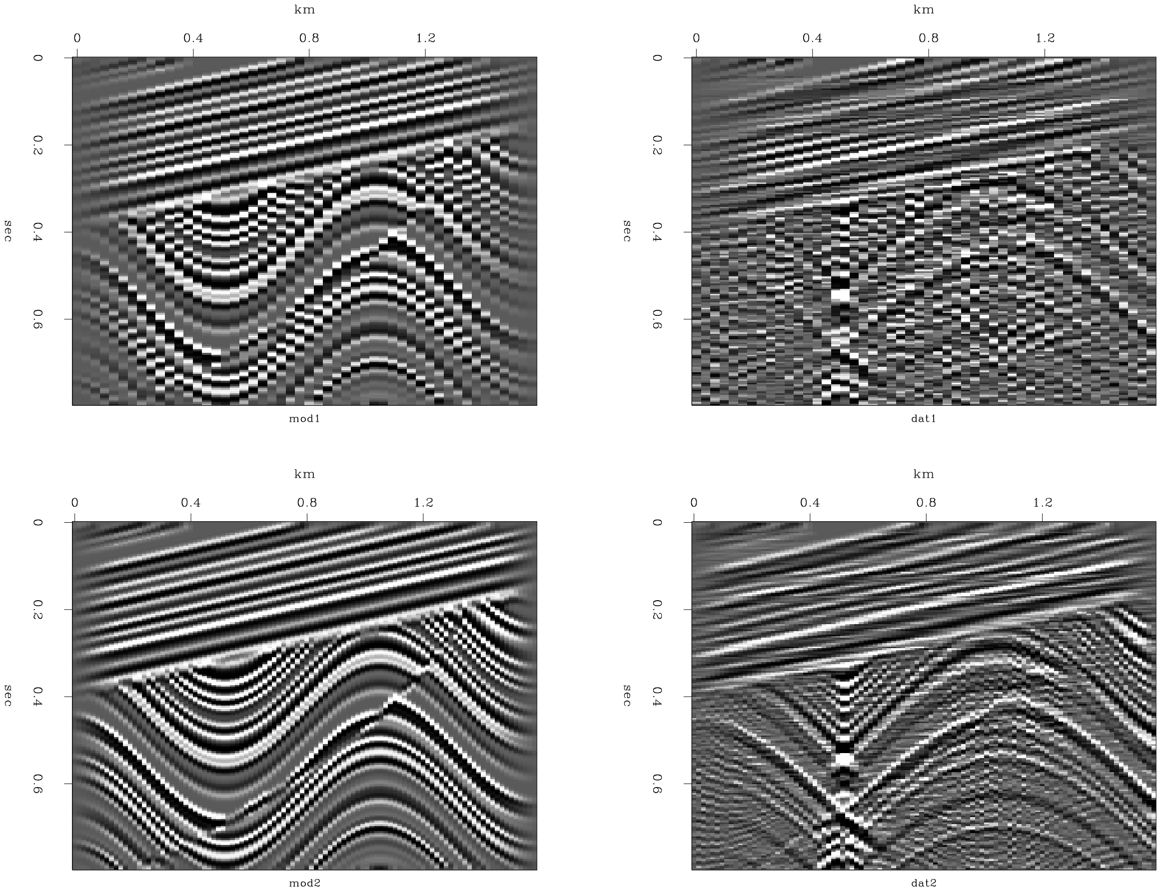

Figure 4: Left is model. Right is synthetic data from the model. Top has 50 points on the x-axis, bottom has 100. jon1.kirch-skmod [ER]