Stanford Exploration Project, Report 72, December 18, 1997, pages 1–39

Stanford Exploration Project, Report 72, December 18, 1997, pages 1–39

Design of inverse Kirchhoff-style filters by LSCG

Jon F. Claerbout[1]

keywords: *two-dimensional, least squares, modeling*

ABSTRACT

Using the least-squares conjugate-gradient methods described in PVI, I devise 2-D inverse filters to a Kirchhoff modeling impulse response. The filter values do not fill a 2-D plane but are constrained to lie along the same trajectory as the impulse response. The technique is also applicable to DMO and residual migration operators. It seems that the cost of computing these operators will be comparable to the usual cost of applying them.

INTRODUCTION

An elementary problem in time-series analysis is to design a one-dimensional filter that is inverse to any specified filter. Such an operation could be done in the Fourier domain were it not for the added requirement that the desired filter be of definite time duration or maybe contain specified gaps. The same concepts apply to the point response of sound in a 2-D earth model. Migration is a form of 2-D filtering where we assume that `crosscorrelation` with the earth's impulse response is a satisfactory approximation to `inverse filtering`. Basic Kirchhoff migration assumes the approximate inverse filter (which in principle is a full 2-D array) vanishes everywhere except along the trajectory given by the wavefront found by ray tracing.

The question of amplitude versus offset (AVO) arises in two ways: first, what is the best AVO on the earth's impulse response? and second, what is the best AVO to put on the migration operator (which should be the inverse operator)? I'm afraid some people assume these two AVO's are the same while in truth the relationship is much more complicated and is more like an inverse relation. These questions may be answered by a careful theoretical study. The same questions arise in dip-moveout analysis, and a variety of different answers have been given by able researchers—it is not an easy problem. The same question again arises in residual depth migration where various techniques such as ray tracing allow a fairly uncomplicated determination of the proper amplitudes on the forward modeled impulse response, but the

[1]**email:** not available

determination of the inverse operator is again not easy.

The desire for correct amplitudes led me to consider 2-D imaging operators in the same mathematical framework as 1-D inverse filters. Such an approach seems hardly controversial from a theoretical point of view. Perhaps it is being ignored because the computability requirements seem wholly unrealistic. I undertook this study to attempt to show that the computability can be kept manageable and to see what kinds of practical gains can arise from designing imaging filters in much the same way as optimum 1-D filters. If this study is successful, it should obviate the need for an oppressive amount of theoretical work, and it should yield better results because many practical factors such as noise and acquisition arrays are readily incorporated.

First notice that the 2-D imaging filter is sparse—it is mostly zeros, clustering around a hyperbola of nonzero values. In this introductory study I choose the trajectory to be hyperbolic. Other trajectories could be more important in practice because a purely numerical approach is more necessary when theoretical approaches are difficult.

The first step is to write a subroutine to convolve two-dimensional filters in some way that avoids multiplying by the many embedded zeros. I started out by representing the filter in a compressed way—without the zeros. The compression is not required for memory saving, but to lay the groundwork for the next stage which is applying the conjugate-gradient subroutines described in my recent book.

It is too early to draw conclusions, but the broad outline of the costs are already apparent. To crosscorrelate a sparse 2-D filter with itself costs the square of the number of nonzero points in it. In practice one such crosscorrelation would add a cost to ordinary migration equal to padding the end of a seismic section by a space equal to the width of the migration operator. This is not a large additional cost. But solving for the inverse filter is an iterative process of steps each as costly as the crosscorrelation. In principle the number of iterations required equals the number of unknowns in the filter, but in practice we usually find that a *much* smaller number is generally satisfactory. By doing only two iterations we should be doing better than conventional processing. Thus the proposed method is very promising. Early examples showed little change after about five iterations. Taking a seismic section about five hyperbola widths wide implies a cost of optimum migration filter design about matching the cost of conventional migration.

PROGRAMS

Subroutine `tothin()` copies the nonzero elements of a two-dimensional array `bb(n1,n2)` to a one-dimensional array `b()`, simultaneously determining it to be of size `nb` and making the two integer-valued subscription arrays `b1(nb)` and `b2(nb)`.

```
# Copy 2-D array into a "thin" 1-D array by omitting zero values.
#
```

```
subroutine tothin( n1,n2,bb,   b1,b2,nb,b)
integer      i1,i2,n1,n2,      nb, b1(nb),b2(nb)
real         bb(n1,n2),        b(nb)
nb = 0
do i2= 1, n2 {
do i1= 1, n1 {
        if( bb(i1,i2) != 0.0 ) { nb = nb+1
                b1( nb) = i1
                b2( nb) = i2
                b ( nb) = bb(i1,i2)
                }
        }}
return; end
```

Subroutine `xthin()` does crosscorrelation and convolution (conjugate pairs) of the "thin" arrays created by `tothin()`. The huge number of arguments arises because instead of characterizing a two-dimensional array as `bb(n1,n2)`, we now characterize that two-dimensional array by `b(nb),b1(nb),b2(nb),bm1,bm2` where two new integer parameters `bm1` and `bm2` define the origin or middle of the array. Given some familiarity with the coding style in PVI, the coding is easy especially since we merely abandon those points that fly beyond the edges of the array.

```
# Crosscorrelate 2-D thin filters  ('m' denotes midpoint)
#
subroutine xthin(conj,bm1,bm2,b1,b2,nb,b,am1,am2,a1,a2,na,a,cm1,cm2,n1,n2,cc)
integer bm1,bm2, ib,nb, b1(nb),b2(nb), ib1,ib2, conj
integer am1,am2, ia,na, a1(na),a2(na), ia1,ia2
integer cm1,cm2,     n1,    n2       , ic1,ic2
real b(nb), a(na), cc(n1,n2)
call conjzero( conj, 0,  na, a,  n1*n2, cc)     # erase output
do ib= 1, nb { ib1 = b1(ib);    ib2 = b2(ib)
do ia= 1, na { ia1 = a1(ia);    ia2 = a2(ia)
        ic1 = cm1 + ib1-bm1 - (ia1-am1)         # Think of these lines as
        ic2 = cm2 + ib2-bm2 - (ia2-am2)         # (ic-cm) = (ib-bm) - (ia-am)
        if( ic1 > 0  &  ic1 <= n1)
        if( ic2 > 0  &  ic2 <= n2)              # Outputs on the mesh?
                if( conj == 0 )
                        cc(ic1,ic2) = cc(ic1,ic2) + a(   ia   ) * b( ib)
                else
                        a(  ia    ) = a(   ia   ) + cc(ic1,ic2) * b( ib)
        }}
return; end
```

The least-squares simultaneous equation solution by conjugate gradients also follows the familiar pattern of PVI.

```
#       least squares fitting of thin kirchoff operator
#
subroutine lskirch(bm1,bm2,b1,b2,nb,b,am1,am2,a1,a2,na,a,rm1,rm2,n1,n2,rr,nitr)
integer bm1,bm2, nb, b1(nb),b2(nb), iter, nitr
integer am1,am2, na, a1(na),a2(na)
```

```
integer rm1,rm2,     n1,    n2
real b(nb), a(na), rr(n1,n2)
temporary real  dr(n1,n2), sr(n1,n2)
temporary real  da(na   ), sa(na   )
call zero( n1*n2, rr);                rr( rm1, rm2) = 1.0
call zero( na, a)
do iter= 0, nitr {
        call xthin(1,bm1,bm2,b1,b2,nb,b, am1,am2,a1,a2,na,da, rm1,rm2,n1,n2,rr)
        call xthin(0,bm1,bm2,b1,b2,nb,b, am1,am2,a1,a2,na,da, rm1,rm2,n1,n2,dr)
        call cgstep( iter, na,a,da,sa,  n1*n2,rr,dr,sr)
        }
return; end
```

After some experimentation I decided to incorporate the possibility of weighting functions. Defining a weighting function requires three more lines of code and applying it in the conjugate gradient code requires another three calls to a vector scaling subroutine. I named the vector scaling `vscale()` and the weighted least-squares fitting subroutine `wlskirch()`.

```
#       c() = scale * a() * b()
#
subroutine vscale( scale, n, a, b, c )
integer i, n
real    scale, a(n), b(n), c(n)
do i= 1, n
        c(i) = scale * a(i) * b(i)
return;  end
```

`wlskirch()` which is littered with debugging snapshots.

```
#       Find WEIGHTED least squares inverse of thin kirchoff-type operator
#
subroutine wlskirch(bm1,bm2,b1,b2,nb,b,am1,am2,a1,a2,na,a,rm1,rm2,n1,n2,rr,ntr)
integer bm1,bm2, nb, b1(nb),b2(nb), iter, ntr
integer am1,am2, na, a1(na),a2(na)
integer rm1,rm2,     n1,    n2,         i1,i2, i
real    b(nb), a(na), rr(n1,n2),        dot, value
temporary real  dr(n1,n2), sr(n1,n2),   ww(n1,n2), wr(n1,n2)
temporary real  da(na   ), sa(na   )
temporary real  seea(n1,n2), seec(n1,n2)
call zero( n1*n2, rr);
                rr( rm1, rm2) = 1.0
do i1= 1, n1 {
do i2= 1, n2 {  ww(i1,i2) = sqrt( 1. + iabs(i1-rm1)  + iabs(i2-rm2) )
                }}
call vscale( 1., n1*n2, ww, rr, wr)
call zero( na, a)
do iter= 0, ntr {
        call vscale( 1., n1*n2, ww, wr, dr)
        call xthin(1,bm1,bm2,b1,b2,nb,b, am1,am2,a1,a2,na,da, rm1,rm2,n1,n2,dr)
        call xthin(0,bm1,bm2,b1,b2,nb,b, am1,am2,a1,a2,na,da, rm1,rm2,n1,n2,dr)
```

```
        call vscale( 1., n1*n2, ww, dr, dr)
        call cgstep( iter, na,a,da,sa,  n1*n2,wr,dr,sr)
######### begin diagnostic output
   if( iter <5 | iter==10 | iter==20 | iter==40 | iter == 80 ){
        value = dot( n1*n2, wr, wr)
        write(7,77) iter, value
        77 format( i3, f15.9)
        call zero( n1*n2, seea)
        do i= 1, nb
                seea( b1(i), b2(i) ) = a(i)
                                        call snap('aa.H', n1,n2, seea)
        call xthin( 0,  bm1,bm2, b1,b2,nb,b,
                        am1,am2, a1,a2,na,a,
                        rm1,rm2, n1,n2,  seec)
                                        call snap('cc.H', n1,n2, seec)
        }
        ######### end diagnostic output
        }
return; end
```

As ever, the main program is a bit of a clutter, though this one less so because the output is all being done from inside the subroutine. I tried to include the `cakefile` but LaTeX insisted on looking for a file named `cakefile.tex`.

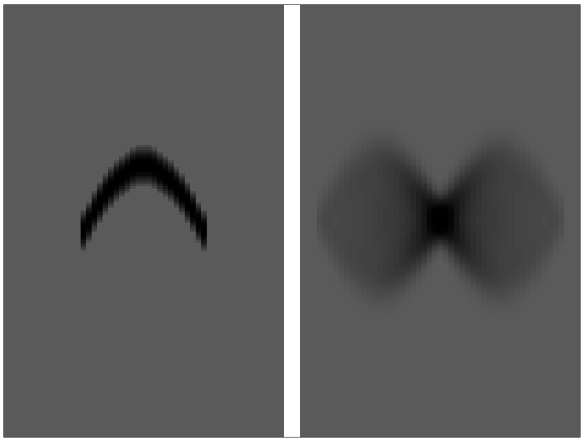

Figure 1: Left is the "Kirchhoff impulse response." Left is also the zero-th iteration of the estimated "time-reversed inverse Kirchhoff impulse response" or for short, "Kirchhoff inverse." Right is the convolution of the two, i.e. the smear function of Kirchhoff modeling and migration. jon1-k0 [ER]

REFERENCES

PVI is an abbreviation for: Claerbout, J.F., 1991, Earth Soundings Analysis : Processing versus Inversion: Blackwell Scientific Publications. Also SEP–**71**.

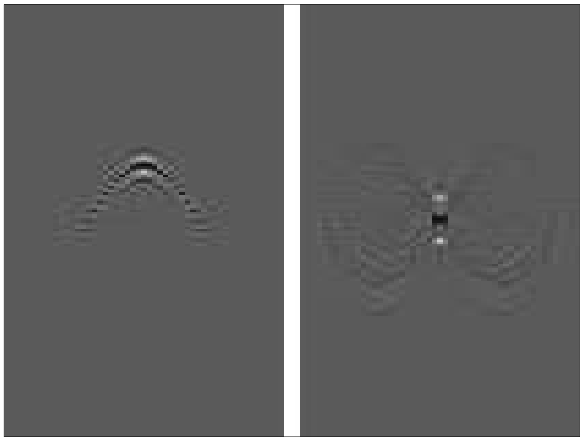

Figure 2: Left is the 80-th iteration of the estimated Kirchhoff inverse. Right is the smear function. Push button for movie over iterations. $\boxed{\text{jon1-k8}}$ [ER]

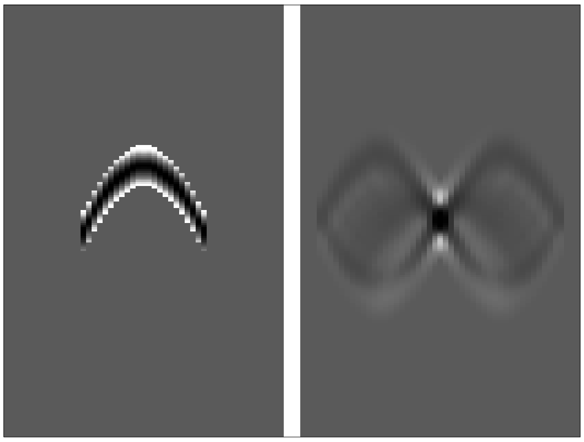

Figure 3: Left is the first iteration of the estimated Kirchhoff inverse. Right is the smear function. $\boxed{\text{jon1-k1}}$ [ER]

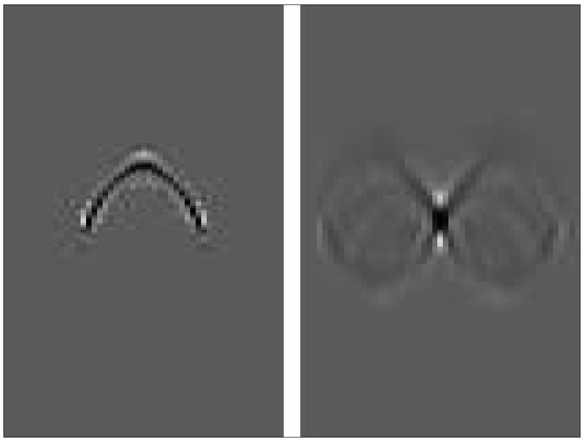

Figure 4: Left is the tenth iteration of the estimated Kirchhoff inverse. Right is the smear function. $\boxed{\text{jon1-k5}}$ [ER]