Stanford Exploration Project, Report 92, November 12, 1997, pages 219–226

Stanford Exploration Project, Report 92, November 12, 1997, pages 219–226

Conjugate-direction Huber regression

Jon Claerbout[1]

keywords: *deconvolution, noise*

ABSTRACT

A straight-forward way to make conjugate-direction regressions robust (insensitive to bursty data noise) is based on the objective function of Huber (least squares for small residuals and least absolute values for large ones). The gradient is based on the clipped residual instead of the residual itself. Likewise the clipped residual is used to define the plane of the gradient and previous step (plane search). This method does not apply to the deconvolution problem because there noisy field data enters into the *operator* for the determination of the prediction-error filter.

INTRODUCTION

The noise variance of data of all kinds is often nonstationary and difficult to characterize. This observation pushes us away from least squares with its requirement that we provide an estimate of the inverse noise variance/covariance, toward robust statistical procedures because they are less sensitive to imperfect knowledge of noise variance.

I seek to provide an electronic textbook with dozens of readily reproducible examples of application of geophysical inverse theory. On my sabbatical leave during the academic year '93-'94, I gathered four wide ranging geophysical data sets, depth sounding (Galilee), radar (Vesuvious), satellite altimetry (Madagasgar), and Seabeam (ocean ridge), and began preparing TDF, "Three dimensional filtering: Environmental soundings image enhancement". The first time I taught this material I discovered that three of the four data sets had problems with severe noise and the fourth had missing values where someone had previously edited out the bad points. Turning a vice into a virtue, I realized that no guide to data fitting is satisfactory without a collection of useful tools for dealing with very noisy data.

I tried to iteratively reweight residuals by the inverse of the observed residual variance. Unfortunately, such reweighting is an art that is difficult to teach, difficult to reuse, and often fails. Iterative reweighting encourages ad-hoc solutions that defy

[1] **email:** jon@sep.stanford.edu

the formal optimization literature. However, iterative reweighting can hardly be avoided since real data is noisy and its variance is usually nonstationary. We should be able to greatly reduce the need for ad hoc reweighting if we use robust methods.

Theoretically, it is well known (FGDP) that the L_1 norm approach allows for a data set containing some infinite data values (noise). A straightforward approach to geophysical inversion (model fitting or regression) is a mixed L_1 and L_2 criterion for fitting data $\mathbf{d}$ by a model $\mathbf{m}$ where

$$\min_{\mathbf{m}} \quad (\sum_{n_d} |\mathbf{F}\mathbf{m} - \mathbf{d}|) + \mathbf{m}'\mathbf{A}'\mathbf{A}\mathbf{m} \tag{1}$$

and where $\mathbf{F}$ links a model to its implied data, and $\mathbf{A}$ is a model roughener. If we had a rapid solution method to equation (1), we would probably use it and be quite happy. Unfortunately, we do not have such a method.

I did some preliminary tests on a regression like equation (1) (but all L_1). I was disappointed that twenty times more iterations than usual were required and each iteration was significantly more costly.[2], From this preliminary experience I sought an approach that would be nearly equal to our customary L_2 approach, but which would adapt itself to a moderate number of large noise values. The gentler approach I adopt here is based on the criteria of Huber (a professor of statistics at Harvard). Residuals are penalized by their square when they are small (L_2) and by their absolute value when they are large (L_1).

We will see that with the Huber approach, the conjugute-direction method is only slightly changed. A big question is whether the convergence rate remains attractive. This is important for geophysical problems which are usually large.

HUBER FUNCTION REGRESSION

I define the Huber function of each residual R as

$$H(R) \quad = \quad \begin{cases} \frac{1}{2}R^2 & \text{for } |R| \le h \\ h|R| - \frac{1}{2}h^2 & \text{for } |R| > h \end{cases} \tag{2}$$

For small residuals R, the Huber function reduces to the usual L_2 least squares penalty function, and for large R it reduces to the usual robust (noise insensitive) L_1 penalty function. Notice the continuity at $|R| = h$ where the Huber function switches from its L_2 range to its L_1 range. Likewise derivatives are continuous at the junctions $|R| = h$:

$$\frac{dH}{dR} \quad = \quad \begin{cases} R & \text{for } |R| \le h \\ h \ \mathrm{sgn}(R) & \text{for } |R| > h \end{cases} \tag{3}$$

[2]I solved L_1 norm multivariate regressions by the conjugate-direction method, doing the plane search by line searching ten times alternately between the gradient and the previous step.

The derivative of the Huber function is what we commonly call the clip function $dH/dR = \mathrm{clip}(R)$.

$$\frac{dH}{dR} \quad = \quad \mathrm{clip}(R) \quad = \quad \begin{cases} -h & \text{for } R < -h \\ R & \text{for } -h \le R \le h \\ h & \text{for } h < R \end{cases} \tag{4}$$

In practice the clip function can be applied at a predetermined value h, or it can be applied at a percentile value of all the R_i.

Now let us set out to minimize a sum of Huber functions of all the components of the residual $\sum_i H(R_i)$ where the residual $\mathbf{R}$ is perturbed by the addition of a small amount of gradient $\mathbf{G}$ and previous step $\mathbf{S}$. The perturbed residual is $\mathbf{R} + \alpha\mathbf{G} + \beta\mathbf{S}$ where we are given $\mathbf{R}$, $\mathbf{G}$, $\mathbf{S}$, and we seek to find α and β by setting to zero derivatives of $\sum_i H(R_i)$ by α and β. For simplicity we assume that α and β are small and that we do not need to worry about components jumping between the L_2 and L_1 range portions of the Huber function. Obviously residual component values will often jump between the two ranges, and because of that, we must iterate the steps I define next:

$$0 \quad = \quad \sum_i \frac{\partial H}{\partial R_i}\frac{\partial R_i}{\partial \alpha} \quad = \quad \sum_{|R_i| \le h} (R_i + \alpha G_i + \beta S_i) G_i \; + \sum_{|R_i| > h} h \ \mathrm{sgn}(R_i) G_i \tag{5}$$

$$0 \quad = \quad \sum_i \frac{\partial H}{\partial R_i}\frac{\partial R_i}{\partial \beta} \quad = \quad \sum_{|R_i| \le h} (R_i + \alpha G_i + \beta S_i) S_i \; + \sum_{|R_i| > h} h \ \mathrm{sgn}(R_i) S_i \tag{6}$$

which reduces to two equations for α and β:

$$\left\{ \sum_{|R_i| \le h} \begin{bmatrix} G_i^2 & G_i S_i \\ G_i S_i & S_i^2 \end{bmatrix} \right\} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad = \quad - \sum_{|R_i| \le h} \begin{bmatrix} R_i G_i \\ R_i S_i \end{bmatrix} - \sum_{|R_i| > h} \begin{bmatrix} h \ \mathrm{sgn}(R_i) G_i \\ h \ \mathrm{sgn}(R_i) S_i \end{bmatrix} \tag{7}$$

$$= \quad - \sum_i \begin{bmatrix} G_i \ \mathrm{clip}(R_i) \\ S_i \ \mathrm{clip}(R_i) \end{bmatrix} \tag{8}$$

The nice thing about the plane search above is that it has an analytic solution given by the inverse of the 2×2 matrix, and this solution is exact unless residual values have hopped between the L_1 and L_2 ranges, which means another iteration is appropriate. To do another iteration, we would update $\mathbf{R} \leftarrow \mathbf{R} + \alpha\mathbf{G} + \beta\mathbf{S}$ and then clip $\mathbf{R}$ another time and repeat the plane search. I hoped the exact solution should be obtained in a finite number of iterations. Lack of numerical precision in my test case made this uncertain.

From the economical viewpoint, whether or not we would iterate for the values of α and β would depend on whether it was costly to compute the new gradient $\mathbf{G} = \mathbf{F}\mathbf{F}'\mathrm{clip}(\mathbf{R})$, that is, whether $\mathbf{F}$ and $\mathbf{F}'$ are costly to apply. If they are, we would want to make sure we got the most value from each $\mathbf{G}$ we had, so we would iterate the plane search for (α, β). Otherwise, if it was cheap to compute the next gradient

$\mathbf{G} = \mathbf{FF}'\text{clip}(\mathbf{R})$, we would do so rather than making the best possible use of the existing gradient (by repeated plane search).

The economical viewpoint may be surpassed by the need to avoid trouble. Limited experiences so far show that instabilities can arise going from one $\mathbf{G}$ to the next. We should be able to control them by iterating to convergence for each $\mathbf{G}$. Failing in that, I believe theory says we are assured stable convergence if we drop back from conjugate directions to steepest descent. All these extra precautions will require more than the straightforward coding below.

Distractions

For simplicity above I omitted the noise variance/covariance. Theoretically, they are easy to include and in practice I would do so. Practically, they are unknown and often erratic so it is a happy coincidence that L_1-nomr reduces the sensitivity to the noise variance.

Useing a preconditioner does not change any of the above conclusions. Then $\mathbf{m} = \mathbf{Bx}$ where $\mathbf{AB} = \mathbf{I}$ so the usual problem $\min_{\mathbf{m}} \left(|\mathbf{Fm} - \mathbf{d}| + |\mathbf{Am}|^2\right)$ becomes simply $\min_{\mathbf{x}} \left(|\mathbf{FBx} - \mathbf{d}| + |\mathbf{x}|^2\right)$

TESTS

I have six simultaneous equations in four unknowns (similar to the `cgmeth()` test in PVI).

```
# matrix multiply and its adjoint
#
subroutine matmul( adj, bb,        nx,x,  ny,y)
integer ix, iy,    adj,            nx,    ny
real                    bb(ny,nx), x(nx), y(ny)
call adjnull(      adj, 0,         x,nx,  y,ny)
do ix= 1, nx {
do iy= 1, ny {
        if( adj == 0 )
                        y(iy) = y(iy) + bb(iy,ix) * x(ix)
        else
                        x(ix) = x(ix) + bb(iy,ix) * y(iy)
        }}
return; end
```

I have a routine to set a clip level and clip the residual.

```
subroutine hclip(  nr, kwant, rr, cr)
integer i,   iter, nr, kwant
real                               rr(nr), cr(nr), huber, cutoff
common huber, cutoff, iter
```

```
#if( iter < 10)
        call quantabs( kwant,nr,rr, cutoff)
do i= 1, nr
        if(      rr(i) >  cutoff ) { cr(i) =  cutoff }
        else if( rr(i) < -cutoff ) { cr(i) = -cutoff }
        else                       { cr(i) =  rr(i) }
return; end
```

From the equations defining the plane search, we recognize the general steps required to convert the usual conjugate-direction solver to a robust one using Huber functions. First, the gradient $\mathbf{g} = \mathbf{F}'\mathrm{clip}(\mathbf{R})$ suggests a direction $\Delta\mathbf{m} = \mathbf{g}$ to move the model. Moving in this direction would cause a residual change of $\Delta\mathbf{R} = \mathbf{G} = \mathbf{F}\Delta\mathbf{m} = \mathbf{F}\mathbf{F}'\mathrm{clip}(\mathbf{R})$. $\mathbf{S}$ is the residual change on the previous iteration. So far, everything is like the usual conjugate-direction method except that the residual $\mathbf{R}$ is clipped.

```
# Conjugate direction descent, minimize  SUM Huber(rr(i))
#            nx
# rr(i)  =  sum aaa(i,j) * x(j)  - yy(i)
#           j=1
subroutine cgmeth( nx,x, nr,yy,rr, aaa, niter, kwant)
integer i, iter,   nx,   nr,            niter, kwant
real    x(nx), yy(nr), rr(nr), aaa(nr,nx), cutoff, huber
common huber, cutoff, iter
temporary real dx(nx), sx(nx), dr(nr), sr(nr),cr(nr)
do i= 1, nx
        x(i) = 0.
do i= 1, nr
        rr(i) = - yy(i)
do iter= 0, niter {
        call hclip( nr, kwant, rr, cr)
        call matmul( 1, aaa, nx,dx, nr,cr)             # dx = A' clip(r)
        call matmul( 0, aaa, nx,dx, nr,dr)             # dr = A dx
        call cdhuber( iter, nx, x,dx,sx,
                            nr,rr,dr,sr,cr)            # x=x+s; rr=rr+ss
        }
return; end
```

The routine work that needs to be done inside each step of any Huberized conjugate direction method is to compute α and β and apply them. This is nothing but our traditional `cgplus()` subroutine with the new equation (8). Starting from the old subroutine `cgplus()` we make a new subroutine `cdhuber()` modifying the old in two places: (1) use the clipped residual to determine the distance, and (2) omit clipped terms from entering the matrix multiplying the vector of (α, β).

```
# A step of conjugate-gradient descent.
#
subroutine cdhuber( iter,   n, x, g, s,   m, rr, gg, ss, cc)
```

```
integer i,          iter,   n,            m
real    x(n), rr(m), g(n), gg(m), s(n), ss(m), cc(m), huber, cutoff
double precision wdot, ddot, sds, gdg, gds, determ, gdr, sdr, alfa, beta
common huber, cutoff            # for printouts
temporary real ww(m)
huber = 0.
do i=1,m
        if( rr(i)==cc(i))
                { ww(i) = 1.; huber= huber + rr(i)*rr(i)/2.             }
        else
                { ww(i) = 0.; huber= huber + rr(i)*cc(i) - cc(i)*cc(i)/2 }
if( iter < 1 ) {                # steepest descent on first iteration.
        do i= 1, n
                s(i) = 0.
        do i= 1, m
                ss(i) = 0.
        if( wdot(m,ww,gg,gg)==0 )   call erexit('cdhuber: first grad vanishes ')
        alfa =  - ddot(m,gg,cc) / wdot(m,ww,gg,gg)
        beta = 0.
        }
else {                          # search plane by solving 2-by-2
        gdg = wdot(m,ww,gg,gg)  #  G . (R + G*alfa + S*beta) = 0
        sds = wdot(m,ww,ss,ss)  #  S . (R + G*alfa + S*beta) = 0
        gds = wdot(m,ww,gg,ss)
        if     ( gdg == 0.) {call seperr('cdhuber: grad vanishes'); return}
        else if( sds == 0.) { alfa= - ddot(m,gg,cc) / wdot(m,ww,gg,gg); beta=0.}
        else { determ = gdg * sds * dmax1( 1.d0 - (gds/gdg)*(gds/sds), 1.0d-12 )
                gdr = - ddot(m,gg,cc)
                sdr = - ddot(m,ss,cc)
                alfa = ( sds * gdr - gds * sdr ) / determ
                beta = (-gds * gdr + gdg * sdr ) / determ
                }
        }
do i= 1, n                                      # s = model step
            s(i)  = alfa * g(i)  + beta * s(i)
do i= 1, m                                      # ss = conjugate
            ss(i) = alfa * gg(i) + beta * ss(i)
do i= 1, n                                      # update solution
            x(i)  =  x(i) +  s(i)
do i= 1, m                                      # update residual
            rr(i) = rr(i) + ss(i)
return; end

double precision function wdot( n, w, x, y )
integer i, n;           real  w(n), x(n), y(n); double precision val
val = 0.;       do i=1,n  { val = val + w(i) * x(i) * y(i) }
wdot = val;     return;         end
```

PREDICTION-ERROR FILTER ESTIMATION

The Huberized conjugate-direction method does not apply to the autoregression (deconvolution) problem because there noisy field data enters into the *operator* for the determination of the prediction-error filter. This eliminates many of the examples in TDF.

I could not resist trying PEF estimation anyway. After all, maybe a bad equation shows up much like a bad data point. It is gratifying that bad regression equations are thrown out of the matrix used to determine (α, β).

Unfortunately, I found the program became extremely unstable, diverging rapidly. To achieve convergence I needed to make the noise small. The filters I got were never close to correct.

It seems the best method to deal with the PEF problem is as currently described in TDF. The way TDF handles such noise is via weighting functions. Generally when computing a prediction-error filter, there are ample equations compared to unknowns. When any regression equation has a large residual, I abandon that equation by applying a zero weight to it. One method is to change the weighting function as the regression proceeds. This is risky. The more reliable method is to solve the problem with fixed weights, then use the residuals to determine new weights to solve the problem a second time.

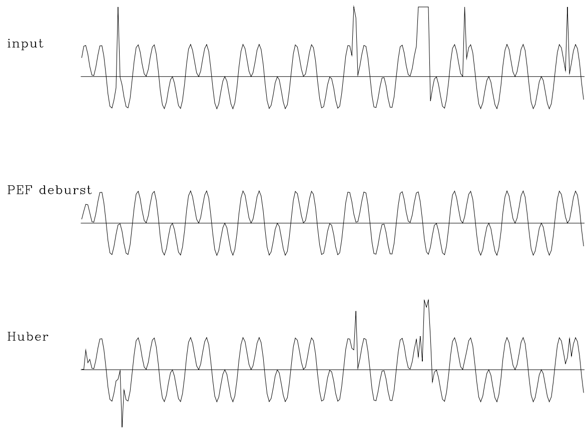

Figure 1: (a) Simple data with bursty noise. (b) Cleaned by L2 method of TDF (c) Huber method. jon2-pefdeburst [ER]

CONCLUSION

I feel the method of this paper did not work very well. I believe I should prepare and test a code for the Huber line search and then use standard conjugate gradients to see if that works better than the conjugate-direction method.