Stanford Exploration Project, Report 92, November 12, 1997, pages 227–235

Stanford Exploration Project, Report 92, November 12, 1997, pages 227–235

Short Note

Medians in multivariate regression

Jon Claerbout[1]

keywords: *modeling, random, optimization*

INTRODUCTION

Here I design filters and models of hyperbola superposition based on medians. Let r_i be a residual and Δr_i be a residual perturbation caused by a change $\Delta\mathbf{m}$ in the model $\mathbf{m}$. With $\mathbf{m} \leftarrow \mathbf{m}+\alpha\Delta\mathbf{m}$, choosing $\alpha = -\text{median}(r_i/\Delta r_r)$ gives us residuals where as many components as possible of the residual are pushed towards zero. This procedure suggests a family of processes that should be immune to bursty noises in data and might quickly give good approximations for inversions of high dimensionality.

MEDIANS AND REGRESSION

Let r_i be components of a residual vector $\mathbf{r}$ and consider perturbation components Δr_i. We study the properties of the new residual

$$\tilde{\mathbf{r}} \quad = \quad \mathbf{r} + \alpha\Delta\mathbf{r} \tag{1}$$

where α is a scaling factor. Consider the implications of

$$0 \quad = \quad \text{median}_i \left(\frac{\tilde{r}_i}{\Delta r_i}\right) \tag{2}$$

$$\quad = \quad \text{median}_i \left(\frac{r_i + \alpha\Delta r_i}{\Delta r_i}\right) = \text{median}_i \left(\frac{r_i}{\Delta r_i} + \alpha\right) = \text{median}_i \left(\frac{r_i}{\Delta r_i}\right) + \alpha \tag{3}$$

Thus choosing $\alpha = -\text{median}_i(r_i/\Delta r_i)$ causes the median of $\tilde{r}_i/\Delta r_i$ to vanish which means $\tilde{r}_i/\Delta r_i$ has as many positive terms as negative ones. Consequently, the number of polarity agreements in the component pairs $(\tilde{r}_i, \Delta r_i)$ of the new residual $\tilde{\mathbf{r}}$ and the regressor $\Delta\mathbf{r}$ equals their polarity disagreements. In other words,

$$0 \quad = \quad \sum_i \text{sgn}(\tilde{r}_i)\,\text{sgn}(\Delta r_i) \quad = \quad \text{agreements} - \text{disagreements} \tag{4}$$

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

In what sense have we created the smallest new residual $\tilde{\mathbf{r}}$? It is smallest in the sense that if we change $\tilde{\mathbf{r}}$ by adding or subtracting some ϵ of the perturbation $\epsilon\Delta\mathbf{r}$, the count of growing components less the decreasing components of $\tilde{\mathbf{r}} + \epsilon\Delta\mathbf{r}$ is

$$\sum_i \operatorname{sgn}(\tilde{r}_i + \epsilon\Delta r_i)\, \operatorname{sgn}(\epsilon\Delta r_i) \quad \geq \quad 0 \tag{5}$$

The best residual is one in which any perturbation serves only to increase the polarity agreements with the regressor. We chose α to reduce as many components of $\tilde{\mathbf{r}}$ as we could.

MEDIANS IN AUTOREGRESSION

Here is an idea for computing autoregression filters. I am not sure what its mathematical properties are, but it seems simpler than other autogression methods and it seems more robust. We load a residual vector with the negative of the data $\mathbf{r} = -\mathbf{d}$ and iterate the following steps:

First take a random direction $\Delta\mathbf{a}$. The residual change connected with going this direction is

$$\Delta\mathbf{r} \quad = \quad \mathbf{D}\,\Delta\mathbf{a} \tag{6}$$

where $\mathbf{D}$ is a convolution matrix (matrix of downshifted columns, each containing the data $\mathbf{d}$). Now we need to decide an amount α of the perturbation $\Delta\mathbf{a}$ to be used for our new filter

$$\mathbf{a} \longleftarrow \mathbf{a} + \alpha\Delta\mathbf{a} \tag{7}$$

We do this by choosing

$$\alpha \quad = \quad -\operatorname{median}_i \left(\frac{r_i}{\Delta r_i} \right) \tag{8}$$

This reduces the maximum number of components in $\mathbf{r}$.

I experimented with a variety of random and nonrandom directions $\Delta\mathbf{a}$. One of my initial guesses $\Delta\mathbf{a} = \operatorname{sgn}(\mathbf{D}')\operatorname{sgn}(\mathbf{r})$ turned out to be a poor one. Better results were found by cycling, one at a time through each filter component. That saves the cost of the correlation too! I also tried random component values in $\Delta\mathbf{a}$. I also tried randomly selecting one component in $\Delta\mathbf{a}$ to be nonzero. None of these methods would tolerate as much noise as the method in TDF. (In that method, a prior filter $(1, -2, 1)$ was guessed, and residuals of it were sorted to point to bursts, which were then weighted by zero.) Some comparisons of the different methods follow:

```
method    niter    a(1)     a(2)       a(3)      a(4)       a(5)
-----     -----   -------  -------   --------  --------   --------
TDF         10   1.00000  -3.44381   4.91496  -3.44027    0.99767
pefcycle  2000   1.00000  -1.54487   0.77143   0.00011   -0.02502
pefrand   2000   1.00000  -1.65394   0.91726   0.03670   -0.09572
pefrandm  2000   1.00000  -1.72055   0.98708   0.00670   -0.11133
```

The TDF method is significantly faster as well as being better since instead of requiring infinitely many iterations, it requires only one iteration for each component of **a**. I am disappointed this median method did not work better than the TDF method because the TDF method requires a prior filter and the median method does not.

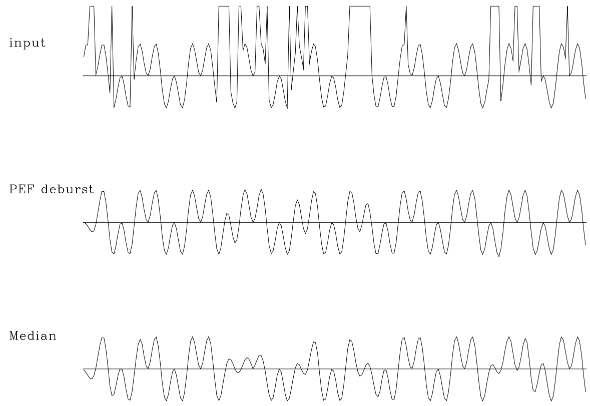

Figure 1: (a) Bichromatic data with bursty noise. (b) Cleaned by L_2 method of TDF. (c) Random directions and median distances. jon3-pefdeburst [ER]

Some comparisons are in Figure 1 which shows, not the autoregressive filters themselves but using the filters, the results of restoration of bad data treating it as missing data. The individual filter coefficients, or random directions, did not work as hoped, but we might have better luck using Levinson's reflection coefficients (not tried).

Let us pursue the idea above that each component of model space can be handled independently, but let us do it in another problem where there are many more components in model space and it seems plausible that many of them would be hardly related. Let us choose a truly huge problem, say inverse hyperbola superposition.

INVERSE HYPERBOLA SUPERPOSITION

Our basic model is

$$\mathbf{d} = \mathbf{H}\mathbf{m} \tag{9}$$

where $\mathbf{m}$ is model space, $\mathbf{d}$ is data space, and $\mathbf{H}$ is a curve superposition such as hyperbola superposition. Obviously we could be talking about velocity analysis or migration and the number of components in model space is huge. These unknown components may be partly correlated, but since there are so many unknowns we will suppose that it is useful to seek an initial guess at a solution by assuming the unknowns are not correlated. To aid in our motivation, we also presume the data contains much bursty noise so the usual ways of doing migration or velocity analysis are in trouble; thus we have a license to seek new ways that are fast and robust.

We begin by loading up a residual vector $\mathbf{r} = -\mathbf{d}$ with the negative of the data. We could start an inverse problem with a gradient, say

$$\Delta\mathbf{m} \quad = \quad \mathbf{H}' \operatorname{sgn}(\mathbf{r}) \tag{10}$$

but here I propose something much simpler. We process only one component of model space. (Later and independently we need to process likewise all other components of model space, and then iterate some more.) Thus $\Delta\mathbf{m}$ has only one nonzero component. We may as well take that component to be unity, say $\Delta m_j = 1$ for some particular j. This perturbation allows us to create a perturbed residual $\Delta\mathbf{r}$. Allow me an approximation of physics and numerical analysis where a single point of model space does not go to every point in data space. In the most elementary modeling procedure the number of points in data space affected by a single point of model space is simply the number of offsets, say n_x. The residual perturbation is

$$\Delta\mathbf{r} \quad = \quad \mathbf{H}\,\Delta\mathbf{m} \tag{11}$$

which gives us a $\Delta\mathbf{r}$ of n_x affected components and the values Δr_i of those components are the values of the amplitude versus offset of the hyperbola in our simple modeling. As before, the new model is $\mathbf{m} + \alpha\Delta\mathbf{m}$, where

$$\alpha \quad = \quad -\ \mathrm{median}_i \left(\frac{r_i}{\Delta r_i} \right) \tag{12}$$

It is fun to compare this iterative process to migration since it is similar, but it leads to a kind of inversion. To begin with, if we had no bursty noise, we could simplify life just a little bit by choosing α to be

$$\alpha \quad = \quad -\ \frac{\sum_i r_i \Delta r_i}{\sum_i \Delta r_i \Delta r_i} \tag{13}$$

This α is the usual normalization that might be used in stacking. Since we work recursively, however, the job of reducing the residual is never done, and the values on the residual plane depend upon the order in which we do the work.

We could choose to do the work in some order to make the code fast. We could also choose to do the work in some "best" order (yet to be defined) for the quality of the results.

There is no need for the data to lie on a regular mesh or for a "complete" data set. Aliasing issues have yet to be examined.

WEIGHTED MEDIANS

The simple median is too simple a concept. We need the weighted median concept to properly allow for the fact that a stacking hyperbola is large near its top and tapers off with distance and angle. A simple median is like a sum over the hyperbola without allowing for amplitude as a function of offset. We need to allow for amplitude with medians too. Weighted medians are explained in FGDP, but I give a short summary here. These two equations are equivalent:

$$0 \;=\; \mathrm{median}_i \left(\frac{r_i}{\Delta r_i} + \alpha \right) \tag{14}$$

$$0 = \frac{\partial}{\partial \alpha} \sum_i \left| \frac{r_i}{\Delta r_i} + \alpha \right| \tag{15}$$

The weighted median with weights Δr_i is defined instead by the equilibrium:

$$0 = \frac{\partial}{\partial \alpha} \sum_i |\Delta r_i| \left| \frac{r_i}{\Delta r_i} + \alpha \right| \tag{16}$$

$$0 = \frac{\partial}{\partial \alpha} \sum_i |r_i + \alpha \Delta r_i| \tag{17}$$

The weighted median minimizes the so-called L_1 norm of the residual thus the weighted median seems a more standard optimization concept. The simple median, not the weighted median however, is more appropriate in the autoregression problem where noise in field data finds its way into the operator. (The data is in the convolution matrix (operator) that multiplies the filter vector.)

DISCUSSION

Paradox

After about 24 hours of contemplating migration with weighted median stacks I began to doubt they would work. The problem is that they do not seem to migrate plane layers! The Fresnel zone is less than half the hyperbola's length. Thus the median seems to be unaffected by the tangency of the hyperbola to the parallel line. This is a paradox that L_1 is convex and yet it seems that a search of a full set of basis functions does not lead us anywhere. Perhaps a "trimmed mean" would do the job.

Decomposition of model space

Weighted medians are equivalent to L_1 optimizations but medians themselves are just an idea that may or may not lead to successful results. In a convex optimization (such as L_1 or L_2) in principle (and in fact when the dimensionality is low) we can find the optimum by blundering about in random directions, as long as we always move to the minimum in each direction. Unweighted medians are not related to convex optimization so they may require the "right" basis.

In the migration problem we are solving an L_1 optimization. We should be able to use random directions, but we could use the gradient. If chosing the "right" direction were important, I would reason that the earth is a sedimentary section and local planes are the proper model space and that we should median-search the data space for the effect of each local plane in model space.

CODE FOR WEIGHTED MEDIAN

I adapted the code below from FGDP to find the weighted median. Theoretically this code requires about three times as many arithmetical operators as a weighted mean. Instead of N multiplications and additions, the weighted median is expected to require about $3N$ multiplications, subtractions, and polarity tests, as well as some lessor number of element interchanges.

If you have the electronic document you can test the code by typing `gmake minabs.test` which uses random numbers for components r_i and Δr_i and compares the minimum produced by the code below to the global search over all $\alpha_j = r_j/\Delta r_j$. for the minimum of $\sum_i |r_i + \alpha_j \Delta r_i|$.

```
#  minimize(alpha)  =   sum  abs( f(i) + w(i)*alpha )
#                        i
subroutine minabs( n, f, w, alpha)
integer i,n,low,large,ml,mh,mlt,mht,itry,j
real gn,gp,t,alpha,er,gnt,gpt,gplx,gmix,small,grad
          real    w(n),f(n)
temporary real    g(n)
temporary integer k(n)
small = 0.
do i=1,n
        k(i) = i
low = 1
large = n
ml = n
mh = 1
gn = 0.
gp = 0.
do itry = 1,n {
        j = k( low+mod( (large-low)/3+itry, large-low+1) )
        if( abs(w(j)) != 0.) {
                t = f(j)/(w(j))
                f(j) = w(j)*t
                do i = low,large {
                        j = k(i)
                        er = f(j)-w(j)*t
                        g(j) = 0.
                        if     ( er>small)
                                        g(j) = -w(j)
                        else if( er<(-small))
                                        g(j) = +w(j)
                        else
                                        g(j) = 0.
                        }
                call split(low,large,k,g,mlt,mht)
                gnt = gn
                do i = low,mlt
                        gnt = gnt+g(k(i))
                gpt = gp
                do i = mht,large
```

```
                        gpt = gpt+g(k(i))
                gplx = 0.
                gmix = 0.
                do i = mlt,mht {
                        j = k(i)
                        if (w(j)<0.) {
                                gplx = gplx-w(j)
                                gmix = gmix+w(j)
                                }
                        if (w(j)>0.) {
                                gplx = gplx+w(j)
                                gmix = gmix-w(j)
                                }
                        }
                grad = gnt+gpt
                if ((grad+gplx)*(grad+gmix)<0.)
                                break 1
                if (grad>=0.)
                                low = mht+1
                if (grad<=0)
                                large = mlt-1
                if (low>large)
                                break 1
                if (grad>=0.)
                                gn = gnt+gmix
                if (grad<=0.)
                                gp = gpt+gplx
                if (grad+gplx==0.)
                                ml = mlt
                }
        if (grad+gmix==0.)
                        mh = mht
        }
alpha = - t
return; end

subroutine split(low,large,k,g,ml,mh)
#       given g(k(i)),i=low,large
#       then rearrange k(i),i=low,large and find ml and mh so that
#       (g(k(i)),i=low,(ml-1)) < 0. and
#       (g(k(i)),i=ml,mh)=0. and
#       (g(k(i)),i=(mh+1),large) > 0.
real    g(1)
integer k(1)
integer low,large,ml,mh,keep,i,ii
ml = low
mh = large
repeat {
        ml = ml-1
        repeat
                ml = ml+1
                until(g(k(ml))>=0)
        repeat {
                mh = mh+1
```

```
                repeat
                        mh = mh-1
                        until(g(k(mh))<=0)
                keep = k(mh)
                k(mh) = k(ml)
                k(ml) = keep
                if (g(k(ml))!=g(k(mh)))
                        break 1         # Break out of enclosing "repeat{"
                do i = ml,mh {
                        ii = i
                        if (g(k(i))!=0.0)
                                go to 30
                        }
                break 2         # Break out of two enclosing "repeat{"'s
                30  keep = k(mh)
                k(mh) = k(ii)
                k(ii) = keep
                }
        }
return
end
```