L_1 Regression Program

Jon F. Claerbout

In 1973 Francis Muir and I published a much-quoted paper on the applicability of the L_1 norm to geophysical problems. Although I developed a program for L_1 multivariate regression, I did not include it in that paper, nor did I include it later in FGDP. The program is given here along with some test cases, and my opinions about the program.

Development of L_1 in Geophysics Since 1973

The L_1 norm continues to be a fascination to researchers. The use of medians and quantiles has become widespread in practice. At the SEP all the plot programs default to quantiles to determine the clip and the CRT plotting programs defaults to an algorithm using quantiles to determine the non-linear compression. We rarely use AGC. Shuki Ronen routinely uses running medians in his statics programs.

So far, the L_1 multivariate regression has received little attention. The only application of L_1 regression in routine practice seems to be for earthquake epicenter locations. Several deconvolution researchers have tried to incorporate L_1 concepts but I don't believe this has been done with L_1 regression. At SEP we make no routine use of L_1 norm regression. This may be because of lack of a compelling application, or it may be lack of a convenient program. So I dug out the 12 year old program listing and reran it. Results are shown in figures 1 and 2.

Degeneracy in the Test Case

The test cases in this study all related to fitting sinusoids to a step function. Curiously, this simple arrangement generates some interesting degeneracies, thus presenting a challenge to all linear programing codes. Ordinarily a certain number of the regression equations are solved exactly at the optimal solution. The number of equations that fit exactly is the same as the number of unknown adjustable parameters. Degeneracy arises when the data is such that more than the required number

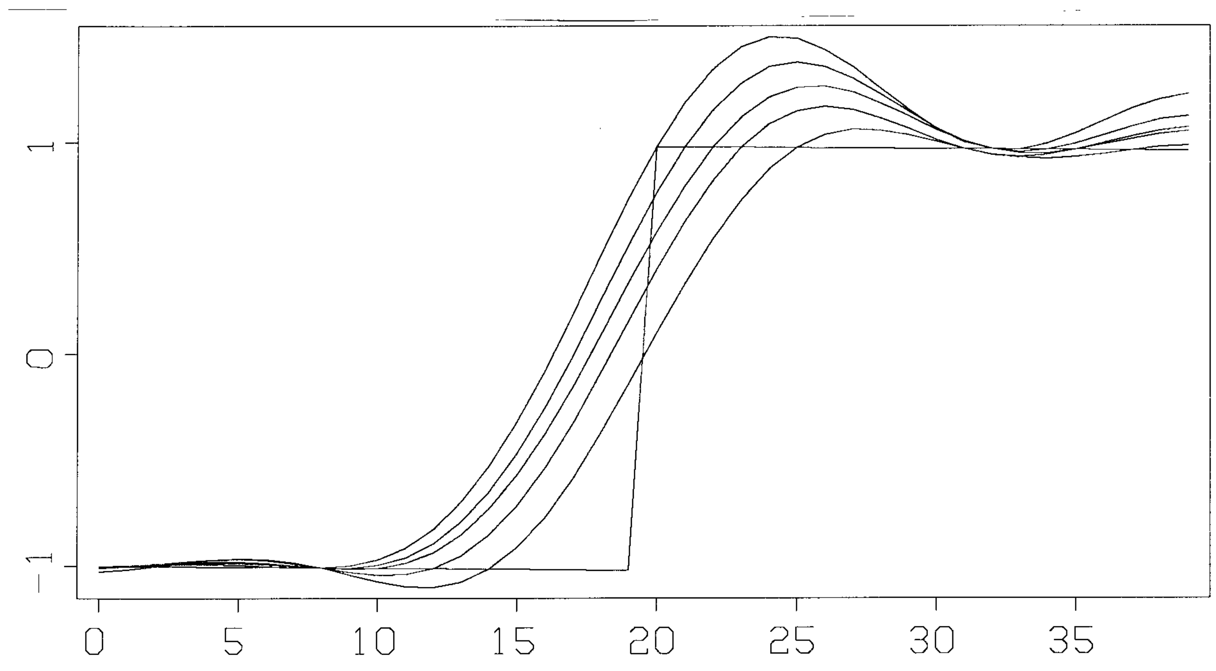


FIG. 1. Fitting a step function with sinusoids. Fits of increasing skewness. (recomputation of figure 9 in Claerbout and Muir [1973])

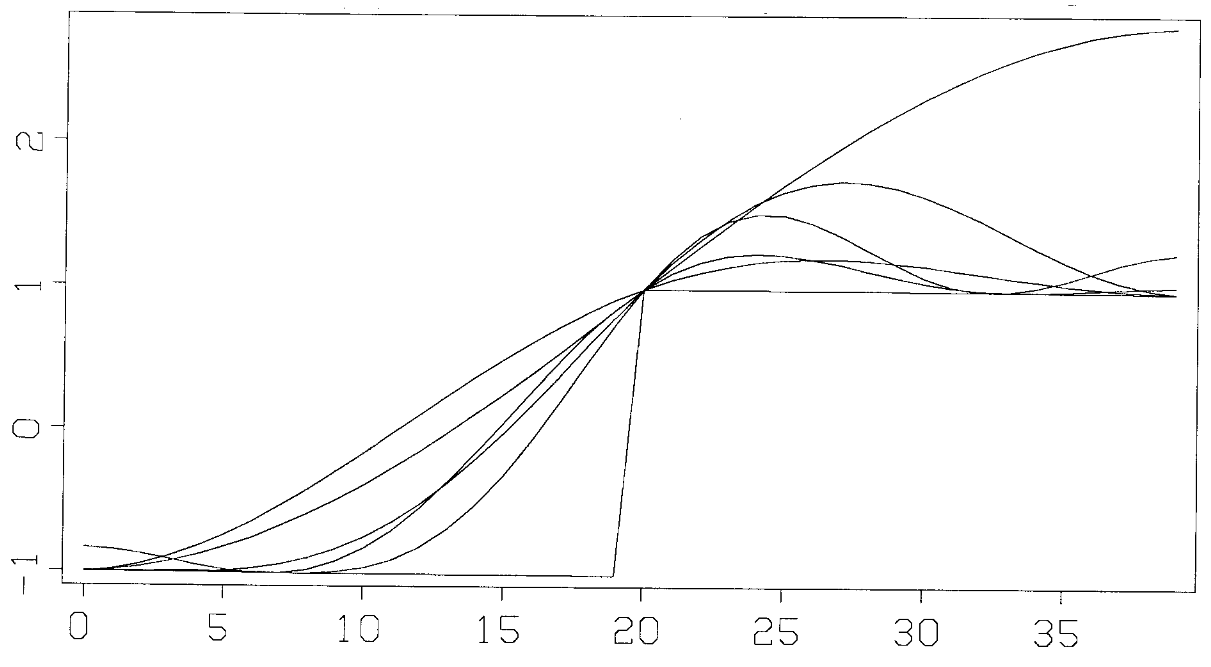


FIG. 2. Upper bounding fit of sinusoids to a step function. Curves of increasing m, the number of sinusoids. The m^{th} order curve touches the step at m places. Fits of increasing skewness. (recomputation of figure 3 in Claerbout and Muir [1973])

of equations is solved. With real, noisy data this wouldn't happen. Here is how it can happen on this synthetic data: A particular sinusoid is the unit constant function. The constant function can fit the step function exactly at half of the time points — thus a massive amount of degeneracy. What happens to the program is that it can get stuck on this solution, and fail to descend all the way. I wasn't interested in this degeneracy but I liked the test case, so I simply added a small slope to the step function.

Program Reliability

One problem in linear programming is deciding whether an equation is satisfied or not. This amounts to deciding whether a number is zero or not. Since single precision accuracy is about six digits, I chose values for practical zero to be less than about 10^{-5}. I had no problems, but the fitting functions were nearly orthogonal in my test cases. Traditional matrix difficulties could arise in other examples. The program is certainly not written to a high standard in numerical analysis. Whether it (or any professionally coded matrix program) is reliable in practice depends mainly on the application to which it is put. In other words, you should try it and see. I have not tried this program with more than about 15 variates, but I think it should work with many more.

The absolute value function is generalized by having a down slope "gd" and an up slope "gu". To impose inequality constraints, you can imagine setting one of the slopes equal to zero and the other slope equal infinity. For my upper bounding fit example I let one slope go to zero and kept the other slope unity. The reason I did this was to maintain the concept that practical zero is 10^{-5}.

Program Listing

The program is given in *Ratfor*, a dialect of Fortran. Most programmers can convert Ratfor to Fortran without any specific instructions. For those with questions, a Ratfor review is in SEP-40 p.67.

REFERENCES

Claerbout, Jon F., and Francis Muir, 1973, *Robust Modeling with Erratic Data,* Geophysics Vol. 38, No. 5, pp 826-844

```
# Claerbout and Muir 73 figure 9
define MAXN1 40              # beware, it is also in subroutine
define MAXN2 20              # beware, it is also in subroutine
real a(MAXN1,MAXN2),x(MAXN2),d(MAXN1),e(MAXN1),gu(MAXN1),gd(MAXN1)
real arg, small
integer n,m,i,j,m,iskew
integer  output, outfd
outfd = output()                        # designate output file
n = 40                                  # n is the length of the step function
m = 5                                   # m is the number of sinusoids.
call fetch("n2 m","i",m)
if( m > MAXN2 )
        call erexit("dimension statement too small")
call putch("title","s","Increasing_Skewness")
call putch ("n1","i",n)
call putch ("n2 or m ","i",m)
call putch ("nplot or n2","i",5+1)
do i = 1,n/2
        d(i) = -1. -.001*i
do i = n/2+1,n
        d(i) = 1. -.001*i
call rite( outfd, d, 4*n )              # output
do iskew = 1,5 {
        do i = 1,n {
                gu(i)=1.
                gd(i)=-1./ 2.**(iskew-1)
                }
        do j = 1,m {
                do i = 1,n {
                        arg = (((i-n/2-.5)*3.14159265*(2*j-2))/(2.*n))
                        if( 2*(j/2) == j )
                                a(i,j) = sin(arg)
                        else
                                a(i,j) = cos(arg)
                        }
                }
        small = 1.e-5
        call elskew(n,m,a,d,gu,gd,small,x)
        do i = 1,n {
                e(i) = 0.
                do j = 1,m
                        e(i) = e(i)+a(i,j)*x(j)
                }
        call rite( outfd, e, 4*n )
        }
stop
end
```

```
# Claerbout and Muir 1973 figure 3
define MAXN1 40                 # beware, it is also in subroutine
define MAXN2 20                 # beware, it is also in subroutine
real a(MAXN1,MAXN2),x(MAXN2),d(MAXN1),e(MAXN1),gu(MAXN1),gd(MAXN1)
real arg, small
integer n,m,i,j,mm
integer  output, outfd
outfd = output()
n = 40
m = 5
call fetch("n2 m","i",m)
if( m > MAXN2 )
          call erexit("dimension statement too small")
call putch("title","s","Upper_bounding_fits")
call putch ("n1","i",n)
call putch ("n2 or m ","i",m)
do i = 1,n {
          gu(i)=1.
          gd(i)=-.0001          # Be more forgiving for negative errors.
          }
do i = 1,n/2
          d(i) = -1. -.001*i
do i = n/2+1,n
          d(i) =  1. -.001*i
call putch ("nplot or n2","i",m)
mm = m
call rite( outfd, d, 4*n )
do m = 2,mm {
          do j = 1,m {
                    do i = 1,n {
                              arg = (((i-n/2-.5)*3.14159265*(2*j-2))/(2.*n))
                              if( 2*(j/2) == j )
                                        a(i,j) = sin(arg)
                              else
                                        a(i,j) = cos(arg)
                              }
                    }
          small = 1.e-5
          call elskew(n,m,a,d,gu,gd,small,x)
          do i = 1,n {
                    e(i) = 0.
                    do j = 1,m
                              e(i) = e(i)+a(i,j)*x(j)
                    }
          call rite( outfd, e, 4*n )
          }
stop
end
```

elskew

```
subroutine elskew(n,m,a,d,gu,gd,small,x)
# find x(i) to minimize
#                    n                    m
#           esum = sum skewnorm(k, sum (d(k)-a(k,i)*x(i)) )
#                  k=1                  i=1
#           skewnorm(k,er) = er*( (if(er>0) gu(k) else gd(k) )
# Jon F. Claerbout 3/73, converted to Ratfor 1/85
real a(n,m),x(m),d(n),gu(n),gd(n)
real sc(20),g(20),gp(20),gm(20),col(20),row(20)
real w(256),f(256),b(20,20)
integer k(256)
integer i,j,l,m,n,ml,mh,loop,kick,new
real hit,wt,oldk,gr,tk,t,esum,small,bigd
if( n>256 ) call err("error exit: n too big")
if( m>20  ) call err("error exit: m too big")
do j = 1,m {
        x(j) = 0.
        do i = 1,m
                b(i,j) = 0.
        sc(j) = 0.
        do i = 1,n
                sc(j) = sc(j)+abs(a(i,j))
        b(j,j) = n /sc(j)
        }
bigd = 0.
do i = 1,n
        if( abs(d(i)) > bigd )
                bigd = abs(d(i))
do i = 1,n {
        f(i) = d(i)   / bigd
        k(i) = i
        }
loop = 0
#           if only gu and gd changed you may reenter here.
#ccccc      entry again
repeat {
        loop = loop+1
        kick = 1+mod(loop-1,m)
        if (loop>m) {
#         find best equation to -kick- out of basis.
                do i = 1,m {
                        gp(i) = 0
                        gm(i) = 0.
                        g(i) = 0.
                        }
                do l = 1,n
                        if (abs(f(l))>=small) {
                                if (f(l)>small)
                                        hit = gu(l)
                                if (f(l)<small)
                                        hit = gd(l)
                                do j = 1,m
                                        g(j) = g(j)-a(l,j)*hit
                                }
                        else
                                do i = 1,m {
                                        wt = 0.
```

...elskew

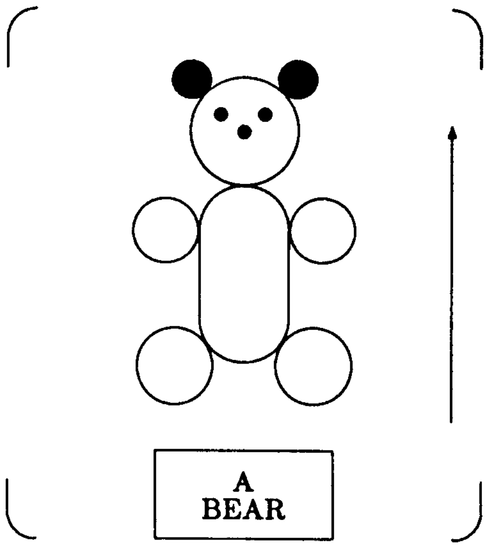
```
                                                do j = 1,m
                                                        wt = wt+a(l,j)*b(j,i)
                                                if (wt<0.)
                                                        gp(i) = gp(i)-gu(l)*wt
                                                if (wt>0.)
                                                        gp(i) = gp(i)-gd(l)*wt
                                                if (wt>0.)
                                                        gm(i) = gm(i)-gu(l)*wt
                                                if (wt<0.)
                                                        gm(i) = gm(i)-gd(l)*wt
                                                }
                        oldk = 0.
                        do i = 1,m {
                                gr = 0
                                do j = 1,m
                                        gr = gr+g(j)*b(j,i)
                                if ((gr+gp(i))*(gr+gm(i))>=0.) {
                                        tk = amin1(abs(gr+gp(i)),abs(gr+gm(i)))
                                        if (tk>oldk)
                                                kick = i
                                        if (tk>oldk)
                                                oldk = tk
                                        }
                                }
                        if (oldk==0.)  # Break out of "repeat { " loop.
                                break 1
                        }
#               find scalar t where x=x0+(col of b)*t
                do i = 1,n {
                        w(i) = 0.
                        do j = 1,m
                                w(i) = w(i)+a(i,j)*b(j,kick)
                        }
                call skewer(n,w,f,gu,gd,small,k,t,ml,mh)
#               write(6,10)(k(i),i = ml,mh)
                10  format(40i3)
#               pick out a new basis equation
                new = k(ml)
                do l = ml,mh
                        if (abs(w(new))<abs(w(k(l))))
                                new = k(l)
#               call putch("kick","i",kick)
#               call putch(" new","i", new)
                t = f(new)/w(new)
                esum = 0
                do i = 1,n {
                        f(i) = f(i)-w(i)*t
                        if (f(i)>0.)
                                esum = esum+gu(i)*f(i)
                        if (f(i)<(-0.))
                                esum = esum+gd(i)*f(i)
                        }
#               update x and basis matrix as described(transposed)in Hadley p.49
                do j = 1,m {
                        col(j) = b(j,kick)
                        x(j) = x(j)+col(j)*t
                        row(j) = 0.
                        do i = 1,m
```

...elskew

```
                    row(j) = row(j)-a(new,i)*b(i,j)/w(new)
               }
          row(kick) = 1./w(new)-1.
          do i = 1,m
               do j = 1,m
                    b(i,j) = b(i,j)+col(i)*row(j)
#              write(6,20) esum, t
          20  format(10e12 .5)
          }
call putch("esum","f",esum)            # Put residual sum into output data base
do j = 1,m
          x(j) = x(j) * bigd
write(6,20) (x(j),j = 1,m)
return
end
```

```
subroutine skewer(n,w,f,gu,gd,small,k,t,ml,mh)
# For a Fortran version all in upper case see FGDP p. 126
#            solve rank 1 overdetermined equations with skew norm
#            inputs- n,w,f,gu,gd,small,k.   outputs- k,t,ml,mh.
#    find t to minimize
#             n
#   ls = sum skewnorm(k,f(k)-w(k)*t)
#            k=1
#   where                       (gu(k)*(er-small) if er>+small gu>0
#   skewnorm(k,er) = (gd(k)*(er+small) if er<-small gd<0
#                               (0.                if abs(er) <= small
#            gu,gd,w,and f are referenced indirectly as w(k(i)),i=1,n etc
#            minima will be at equations k(ml),k(ml+1),...k(mh).
real w(n),f(n),gu(n),gd(n),g(100)
integer k(n)
integer i,n,low,large,ml,mh,mlt,mht,itry,l
real gn,gp,t,er,gnt,gpt,gplx,gmix,small,grad
low = 1
large = n
ml = n
mh = 1
gn = 0.
gp = 0.
do itry = 1,n {
          l = k(low+mod((large-low)/3+itry,large-low+1))
          if (abs(w(l))!=0.) {                    # "!=" means "not equal"
                    t = f(l)/(w(l))
                    f(l) = w(l)*t
                    do i = low,large {
                              l = k(i)
                              er = f(l)-w(l)*t
                              g(l) = 0.
                              if (er>small)
                                        g(l) = -w(l)*gu(l)
                              if (er<(-small))
                                        g(l) = -w(l)*gd(l)
                              }
                    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 {
                              l = k(i)
                              if (w(l)<0.) {
                                        gplx = gplx-w(l)*gu(l)
                                        gmix = gmix-w(l)*gd(l)
                                        }
                              if (w(l)>0.) {
                                        gplx = gplx-w(l)*gd(l)
                                        gmix = gmix-w(l)*gu(l)
                                        }
                              }
                    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
            }
ml = min0(ml,mlt)
mh = max0(mh,mht)
return
end
```

```
subroutine split(low,large,k,g,ml,mh)
# For a Fortran version all in upper case, see FGDP p 127
#           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(100)
integer k(100)
integer low,large,k,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
```



```
\begin{center}
\setlength{\unitlength}{1em}
\begin{picture}(20,20)
\put(10,10){\oval(3,6)}
\put(7.65,6.9){\circle{2.8}}
\put(12.35,6.9){\circle{2.8}}
\put(7.35,11.5){\circle{2.5}}
\put(12.65,11.5){\circle{2.5}}
\put(10,14.85){\circle{4}}
\put(10,14.8){\circle*{.5}}
\put(9.2,15.4){\circle*{.5}}
\put(10.7,15.4){\circle*{.5}}
\put(8.2,16.6){\circle*{2}}
\put(11.8,16.6){\circle*{2}}
\put(17,5){\vector(0,1){10}}
\put(3,3){\oval(2,4)[bl]}
\put(3,17){\oval(2,4)[tl]}
\put(17,17){\oval(2,4)[tr]}
\put(17,3){\oval(2,4)[br]}
\put(7,1){\framebox(6,3){\shortstack{A\\BEAR}}}
\end{picture}
\end{center}
```

Figure 1: Picture Example