Locally Weighted Linear Regression in LOWESS: Cleveland's Method

R.E. Deakin[1] and M.N. Hunter[2]
[1]Dunsborough, WA, 6281, Australia; [2]Maribyrnong, VIC, 3032, Australia
email: randm.deakin@gmail.com
12-Jun-2020

Abstract

Lowess (locally weighted scatterplot smoothing) is a robust weighted regression smoothing algorithm introduced by William S. Cleveland in 1979. In 1981 Cleveland made available FORTRAN routines LOWESS and LOWEST from the Computing Information Library at Bell Laboratories. These are reproduced in the Appendix. The routine LOWEST performs a locally weighted least squares linear regression on a set of data pairs $\left(x_j, y_j\right)$ $j = 1,2,3,\ldots,q$ where the weights are functions of the distances r_j from the point to be 'smoothed' $\left(x_s, y_s\right)$. The routine returns the estimate $\hat{y}_s = \beta_0 + \beta_1 x_s$ where β_0, β_1 are the parameters of a line of best fit where the x_j are considered error-free.

Routine LOWEST uses a clever modification of the usual weighted least squares regression which will be explained below.

Introduction

Lowess (locally weighted scatterplot smoothing) is a robust weighted regression smoothing algorithm proposed by William S. Cleveland (Cleveland 1979). For n data pairs $\left(x_i, y_i\right)$ $i = 1,2,\ldots,n$ where the x-values are considered as independent and error-free and the y-values as measurements subject to error, the algorithm assumes the n points are ordered from smallest to largest x-value and selects a smoothing point, say $\left(x_s, y_s\right)$ $s = 1,2,\ldots,n$ and its q nearest neighbours, noting that the smoothing point $\left(x_s, y_s\right)$ is a neighbour of itself. These q nearest neighbours are a subset of the n data pairs and the algorithm fits a polynomial to the subset that is used to calculate the estimate $\left(x_s, \hat{y}_s\right)$ noting that the 'hat' symbol $\left(\hat{\ }\right)$ denotes an estimate of a quantity. Cleveland (1979, p. 833) suggests that polynomials of degree 1: $y = \beta_0 + \beta_1 x$ (a straight line) or degree 2: $y = \beta_0 + \beta_1 x + \beta_2 x^2$ (a quadratic curve) are sufficient for most purposes and notes that the polynomial of degree 1 "should almost always provide adequate smoothed points and computational ease." In this paper we only consider polynomials of degree 1. Now, since only two points are required to define a straight line, and q will always be greater than 2 in practice, *least squares* is used to determine estimates of the parameters of the *line of best fit* with *local weights* $0 \le w_j \le 1$ for $j = 1,2,\ldots,q$ as functions of the distances from the smoothing point $\left(x_s, y_s\right)$ to each of the q nearest neighbours. [The weight function most often used in lowess smoothing is known as *tricube* (more about this later) and yields local weights that decrease from 1 at the smoothing point to 0 at the furthest of the q points.] After computing the estimate $\hat{y}_s$ at the smoothing point from $\hat{y}_s = \beta_0 + \beta_1 x_s$ (using locally weighted linear regression) the smoothing point is increased by one, i.e., $s = s + 1$ and the subset of q nearest neighbours determined (which may be the same subset as for the previous smoothing point) and the next estimate computed. This process is repeated until $s = n$

Least Squares Linear Regression

The y-values in the $\left(x_j, y_j\right)$ data pairs are assumed to be measurements subject to error and if blunders and systematic errors are eliminated, the remaining random errors can be allowed for by the application of small corrections known as *residuals.* Hence, we write

$$\text{measurement} + \text{residual} = \text{best estimate} \tag{1}$$

Also, a quantity that is being measured has both a true value (forever unknown) and an estimated value (the best estimate) and after removing blunders and systematic errors from the measurements leaving only random errors of measurements, we may write

$$\text{measurement} = \text{true value} + \text{random error}$$

Often, a measurement may be the mean of several measurements or measurements may be obtained from different types of equipment or measurement processes and they may be of varying precision. To allow for this we may weight our measurements, where a *weight* is a numerical value that reflects the degree of confidence we have in the measurement. The greater the weight the more confident we are in the particular measurement. A weight is often defined to be inversely proportional to the *variance* of a measurement where variance is a statistical measure of *precision*. Precise measurements have a small variance.

To solve for the values of the two parameters β_0, β_1 we write q observation equations having the general form of (1)

$$y_j + v_j = \hat{y}_j \quad \text{or} \quad v_j = \hat{y}_j - y_j \tag{2}$$

where v_j denotes the residual of the j^{th} point and $\hat{y}_j$ denotes the best estimate.

Now the *least squares principle* is that the best estimates are those that make the sum of the squares of the residuals, multiplied by their weights, a minimum. To achieve this, write the *least squares function* φ as

$$\varphi = w_1 v_1^2 + w_2 v_2^2 + \cdots + w_n v_n^2 = \sum w_j v_j^2 = \sum w_j \left(\hat{y}_j - y_j\right)^2$$

where the following summation notations are equivalent: $\sum v_j = \sum_j v_j = \sum_{j=1}^{q} v_j = v_1 + v_2 + v_3 + \cdots + v_q$

And since $\hat{y}_j = \beta_0 + \beta_1 x_j$

$$\varphi = \varphi\left(\beta_0, \beta_1\right) = \sum w_j \left(\beta_0 + \beta_1 x_j - y_j\right)^2$$

$\varphi\left(\beta_0, \beta_1\right)$ will have a minimum value when the partial derivatives $\frac{\partial \varphi}{\partial \beta_0}, \frac{\partial \varphi}{\partial \beta_1}$ both equal zero, that is when

$$\begin{aligned} \frac{\partial \varphi}{\partial \beta_0} &= 2\sum w_j \left(\beta_0 + \beta_1 x_j - y_j\right) = 0 \\ \frac{\partial \varphi}{\partial \beta_1} &= 2\sum w_j x_j \left(\beta_0 + \beta_1 x_j - y_j\right) = 0 \end{aligned} \tag{3}$$

and cancelling the 2's in (3) and rearranging gives two *normal equations*

$$\begin{aligned} \left(\sum w_j\right)\beta_0 + \left(\sum w_j x_j\right)\beta_1 &= \sum w_j y_j \\ \left(\sum w_j x_j\right)\beta_0 + \left(\sum w_j x_j^2\right)\beta_1 &= \sum w_j x_j y_j \end{aligned} \tag{4}$$

The solutions of the normal equations (4) give

$$\beta_0 = \frac{\sum w_j x_j^2 \sum w_j y_j - \sum w_j x_j \sum w_j x_j y_j}{\sum w_j \sum w_j x_j^2 - \left(\sum w_j x_j\right)^2} \qquad \beta_1 = \frac{\sum w_j \sum w_j x_j y_j - \sum w_j x_j \sum w_j y_j}{\sum w_j \sum w_j x_j^2 - \left(\sum w_j x_j\right)^2} \tag{5}$$

Now having determined β_0, β_1 the estimates are $\hat{y}_j = \beta_0 + \beta_1 x_j$. This is the typical method least squares linear regression.

Cleveland's Method

Cleveland (1981) gave a very brief outline of his method of scatterplot smoothing and then gave instructions on obtaining FORTRAN routines LOWESS and LOWEST from the Computing Information Library at Bell Laboratories. The routine LOWESS, which is directly called by the user calls a support routine LOWEST and it is withing this support routine that a very efficient and clever weighted least squares regression is employed. The documentation and Ratfor[1] versions of LOWESS and LOWEST are shown in the Appendix and it is subroutine LOWEST that actually computes the least squares estimate at the smoothing point.

Consider the normal equations (4) for a weighted least squares solution for the parameters β_0, β_1 of the regression line (line of best fit) $y = \beta_0 + \beta_1 x$ for the data pairs $\left(x_j, y_j\right)$ with weights w_j for $j = 1,2,3,\ldots,q$.

These equations may be written in terms of <u>normalized weights</u> w_j^* and <u>reduced coordinates</u> $\overline{x}_j$ defined as

$$w_j^* = \frac{w_j}{\sum w_j} \tag{6}$$

$$\overline{x}_j = x_j - g \tag{7}$$

where $g = \dfrac{\sum w_j^* x_j}{\sum w_j^*}$ is a weighted mean, and the normal equations (4) can be written as

$$\begin{aligned} \left(\sum w_j^*\right)\beta_0 + \left(\sum w_j^* \overline{x}_j\right)\beta_1 &= \sum w_j^* y_j \\ \left(\sum w_j^* \overline{x}_j\right)\beta_0 + \left(\sum w_j^* \overline{x}_j^2\right)\beta_1 &= \sum w_j^* \overline{x}_j y_j \end{aligned} \tag{8}$$

We now show that (i) $\sum w_j^* = 1$ and (ii) $\sum w_j^* \overline{x}_j = 0$.

(i) Since $w_j^* = \dfrac{w_j}{\sum w_j}$ then $\sum w_j^* = \dfrac{w_1}{\sum w_j} + \dfrac{w_2}{\sum w_j} + \cdots + \dfrac{w_n}{\sum w_j} = \dfrac{w_1 + w_2 + \cdots + w_q}{\sum w_j} = \dfrac{\sum w_j}{\sum w_j} = 1$

(ii) Since $g = \dfrac{\sum w_j^* x_j}{\sum w_j^*}$ and $\sum w_j^* = 1$ then $g = \sum w_j^* x_j$. Also, $w_j^* \overline{x}_j = w_j^*\left(x_j - g\right) = w_j^* x_j - w_j^* g$.

So $\sum w_j^* \overline{x}_j = \sum w_j^* x_j - w_j^* g = \sum w_j^* x_j - g \sum w_j^* = g - g = 0$

Using these results in (8) gives the solutions

$$\beta_0 = \sum w_j^* y_j \quad \text{and} \quad \beta_1 = \frac{\sum w_j^* \overline{x}_j y_j}{\sum w_j^* \overline{x}_j^2} \tag{9}$$

For the smoothing point $\left(x_s, y_s\right)$ the estimate $\hat{y}_s = \beta_0 + \beta_1 \overline{x}_s$ and using (9) we may write

$$\hat{y}_s = \sum w_j^* y_j + \overline{x}_s \frac{\sum w_j^* \overline{x}_j y_j}{\sum w_j^* \overline{x}_j^2} = \sum w_j^* y_j + \left(\frac{\overline{x}_s}{\sum w_j^* \overline{x}_j^2}\right) \sum w_j^* \overline{x}_j y_j \tag{10}$$

Let $b = \dfrac{\overline{x}_s}{\sum w_j^* \overline{x}_j^2}$ then (10) becomes

[1] Ratfor (short for *Rational Fortran*) is a programming language implemented as a pre-processor for Fortran 66. It provided modern control structures, unavailable in Fortran 66, to replace GOTOs and statement numbers (Wikipedia).

$$\begin{aligned}\hat{y}_s &= \sum w_j^* y_j + b\sum w_j^* \overline{x}_j y_j \\ &= w_1^* y_1 + w_2^* y_2 + \cdots + w_q^* y_q + b\left(w_1^* \overline{x}_1 y_1 + w_2^* \overline{x}_2 y_2 + \cdots + w_q^* \overline{x}_q y_q\right) \\ &= y_1\left(w_1^* + b w_1^* \overline{x}_1\right) + y_2\left(w_2^* + b w_2^* \overline{x}_2\right) + \cdots + y_q\left(w_q^* + b w_q^* \overline{x}_q\right) \\ &= w_1^*\left(1 + b\overline{x}_1\right) y_1 + w_2^*\left(1 + b\overline{x}_2\right) y_2 + \cdots + w_q^*\left(1 + b\overline{x}_q\right) y_q \qquad (11)\end{aligned}$$

And with the substitution $W_j = w_j^*\left(1 + b\overline{x}_j\right)$ in (11) the estimate at the smoothing point $\left(x_s, y_s\right)$ is given by

$$\hat{y}_s = W_1 y_1 + W_2 y_2 + \cdots + W_q y_q = \sum_{j=1}^{q} W_j y_j \qquad (12)$$

You can see the application of Cleveland's least squares method in the Ratfor code for the FORTRAN subroutine LOWEST, shown in the Appendix lines 245 to 348. In particular (i) local weights are calculated and their sum obtained in lines 311-321; weights are normalized in a do loop in lines 326-7; a weighted mean is calculated in a do loop in lines 329-331; a reduced x-coordinate for the smoothing point is calculated in line 332; the factor $b = \dfrac{\overline{x}_s}{\sum w_j^* \overline{x}_j^2}$ is calculated in line 338; the modified weights $W_j = w_j^*\left(1 + b\overline{x}_j\right)$ are calculated in a do loop lines 339-340; and finally the estimate at the smoothing point is calculated from (12) in lines 343-345.

The local weights in subroutine LOWEST are computed from a *tricube weight function* $w_j = \left(1 - \left(\dfrac{r_j}{h}\right)^3\right)^3$

where r_j is the absolute vale of the x-distance from the smoothing point to the j^{th} nearest neighbour and $h = \max\left(r_j\right)$. The weights vary from 1 at the smoothing point where $r_j = 0$ to zero at the point furthest from the smoothing point where $r_j = h$. The calculation of these local weights are shown in lines 307-321.

References

Cleveland, W.S., (1979), 'Robust locally weighted regression and smoothing scatterplots', *Journal of the American Statistical Association,* Vol. 74, No. 368 (Dec., 1979), pp. 829-836 http://home.eng.iastate.edu/~shermanp/STAT447/Lectures/Cleveland%20paper.pdf [accessed 23 Sep 2019]

Cleveland, W.S., (1981), 'LOWESS: A program for smoothing scatterplots by robust locally weighted regression', *The American Statistician,* Vol. 35, No. 1 (Feb., 1981), p. 54

Appendix

FORTRAN program LOWESS

https://github.com/andreas-h/pyloess/blob/master/src/lowess.f

```
* wsc@research.bell-labs.com Mon Dec 30 16:55 EST 1985
* W. S. Cleveland
* Bell Laboratories
* Murray Hill NJ 07974
*
* outline of this file:
*    lines 1-72   introduction
*        73-177   documentation for lowess
*       178-238   ratfor version of lowess
*       239-301   documentation for lowest
*       302-350   ratfor version of lowest
*       351-end   test driver and fortran version of lowess and lowest
*
*   a multivariate version is available by "send dloess from a"
*
*              COMPUTER PROGRAMS FOR LOCALLY WEIGHTED REGRESSION
*
*             This package consists  of  two  FORTRAN  programs  for
*        smoothing    scatterplots   by   robust   locally   weighted
*        regression, or lowess.   The  principal  routine  is  LOWESS
*        which   computes   the  smoothed  values  using  the  method
*        described in The Elements of Graphing Data, by William S.
*        Cleveland    (Wadsworth,    555 Morego   Street,   Monterey,
*        California 93940).
*
*             LOWESS calls a support routine, LOWEST, the code for
*        which is included. LOWESS also calls a routine  SORT,  which
*        the user must provide.
*
*             To reduce the computations, LOWESS  requires  that  the
*        arrays  X  and  Y,  which  are  the  horizontal and vertical
*        coordinates, respectively, of the scatterplot, be such  that
*        X  is  sorted  from  smallest  to  largest.   The  user must
*        therefore use another sort routine which will sort X  and  Y
*        according  to X.
*             To summarize the scatterplot, YS,  the  fitted  values,
*        should  be  plotted  against X.   No  graphics  routines are
*        available in the package and must be supplied by the user.
*
*             The FORTRAN code for the routines LOWESS and LOWEST has
*        been   generated   from   higher   level   RATFOR   programs
*        (B. W. Kernighan, ``RATFOR:  A Preprocessor for  a  Rational
*        Fortran,''  Software Practice and Experience, Vol. 5 (1975),
*        which are also included.
*
*             The following are data and output from LOWESS that  can
*        be  used  to check your implementation of the routines.  The
*        notation (10)v means 10 values of v.
*
```

```
*
*
*
*        X values:
*          1  2  3  4  5  (10)6  8  10  12  14  50
*
*        Y values:
*           18  2  15  6  10  4  16  11  7  3  14  17  20  12  9  13  1  8  5  19
*
*
*        YS values with F = .25, NSTEPS = 0, DELTA = 0.0
*         13.659  11.145  8.701  9.722  10.000  (10)11.300  13.000  6.440  5.596
*           5.456  18.998
*
*        YS values with F = .25, NSTEPS = 0 ,  DELTA = 3.0
*          13.659  12.347  11.034  9.722  10.511  (10)11.300  13.000  6.440  5.596
*            5.456  18.998
*
*        YS values with F = .25, NSTEPS = 2, DELTA = 0.0
*          14.811  12.115  8.984  9.676  10.000  (10)11.346  13.000  6.734  5.744
*            5.415  18.998
*
*
*
*
*                                   LOWESS
*
*
*
*        Calling sequence
*
*        CALL LOWESS(X,Y,N,F,NSTEPS,DELTA,YS,RW,RES)
*
*        Purpose
*
*        LOWESS computes the smooth of a scatterplot of Y  against  X
*        using  robust  locally  weighted regression.  Fitted values,
*        YS, are computed at each of the  values  of  the  horizontal
*        axis in X.
*
*        Argument description
*
*              X = Input; abscissas of the points on the
*                  scatterplot; the values in X must be ordered
*                  from smallest to largest.
*              Y = Input; ordinates of the points on the
*                  scatterplot.
*              N = Input; dimension of X,Y,YS,RW, and RES.
*              F = Input; specifies the amount of smoothing; F is
*                  the fraction of points used to compute each
*                  fitted value; as F increases the smoothed values
*                  become smoother; choosing F in the range .2 to
*                  .8 usually results in a good fit; if you have no
*                  idea which value to use, try F = .5.
*         NSTEPS = Input; the number of iterations in the robust
*                  fit; if NSTEPS = 0, the nonrobust fit is
*                  returned; setting NSTEPS equal to 2 should serve
```

```
*                  most purposes.
*          DELTA = input; nonnegative parameter which may be used
*                  to save computations; if N is less than 100, set
*                  DELTA equal to 0.0; if N is greater than 100 you
*                  should find out how DELTA works by reading the
*                  additional instructions section.
*             YS = Output; fitted values; YS(I) is the fitted value
*                  at X(I); to summarize the scatterplot, YS(I)
*                  should be plotted against X(I).
*             RW = Output; robustness weights; RW(I) is the weight
*                  given to the point (X(I),Y(I)); if NSTEPS = 0,
*                  RW is not used.
*            RES = Output; residuals; RES(I) = Y(I)-YS(I).
*
*
*        Other programs called
*
*               LOWEST
*               SSORT
*
*        Additional instructions
*
*        DELTA can be used to save computations.   Very  roughly  the
*        algorithm  is  this:   on the initial fit and on each of the
*        NSTEPS iterations locally weighted regression fitted  values
*        are computed at points in X which are spaced, roughly, DELTA
*        apart; then the fitted values at the  remaining  points  are
*        computed  using  linear  interpolation.   The  first locally
*        weighted regression (l.w.r.) computation is carried  out  at
*        X(1)  and  the  last  is  carried  out at X(N).  Suppose the
*        l.w.r. computation is carried out at  X(I).   If  X(I+1)  is
*        greater  than  or  equal  to  X(I)+DELTA,  the  next  l.w.r.
*        computation is carried out at X(I+1).   If  X(I+1)  is  less
*        than X(I)+DELTA, the next l.w.r.  computation is carried out
*        at the largest X(J) which is greater than or equal  to  X(I)
*        but  is not greater than X(I)+DELTA.  Then the fitted values
*        for X(K) between X(I)  and  X(J),  if  there  are  any,  are
*        computed  by  linear  interpolation  of the fitted values at
*        X(I) and X(J).  If N is less than 100 then DELTA can be  set
*        to  0.0  since  the  computation time will not be too great.
*        For larger N it is typically not necessary to carry out  the
*        l.w.r.  computation for all points, so that much computation
*        time can be saved by taking DELTA to be  greater  than  0.0.
*        If  DELTA =  Range  (X)/k  then,  if  the  values  in X were
*        uniformly  scattered  over  the  range,  the   full   l.w.r.
*        computation  would be carried out at approximately k points.
*        Taking k to be 50 often works well.
*
*        Method
*
*        The fitted values are computed by using the nearest neighbor
*        routine  and  robust locally weighted regression of degree 1
*        with the tricube weight function.  A few additional features
*        have  been  added.  Suppose r is FN truncated to an integer.
*        Let  h  be  the  distance  to  the  r-th  nearest   neighbor
*        from X(I).   All  points within h of X(I) are used.  Thus if
*        the r-th nearest neighbor is exactly the  same  distance  as
```

```
*        other  points,  more  than r points can possibly be used for
*        the smooth at  X(I).   There  are  two  cases  where  robust
*        locally  weighted regression of degree 0 is actually used at
*        X(I).  One case occurs when  h  is  0.0.   The  second  case
*        occurs  when  the  weighted  standard error of the X(I) with
*        respect to the weights w(j) is  less  than  .001  times  the
*        range  of the X(I), where w(j) is the weight assigned to the
*        j-th point of X (the tricube  weight  times  the  robustness
*        weight)  divided by the sum of all of the weights.  Finally,
*        if the w(j) are all zero for the smooth at X(I), the  fitted
*        value is taken to be Y(I).
*
*
*
*
*  subroutine lowess(x,y,n,f,nsteps,delta,ys,rw,res)
*  real x(n),y(n),ys(n),rw(n),res(n)
*  logical ok
*  if (n<2){ ys(1) = y(1); return }
*  ns = max0(min0(ifix(f*float(n)),n),2)  # at least two, at most n points
*  for(iter=1; iter<=nsteps+1; iter=iter+1){      # robustness iterations
*        nleft = 1; nright = ns
*        last = 0        # index of prev estimated point
*        i = 1   # index of current point
*        repeat{
*                while(nright<n){
*  # move nleft, nright to right if radius decreases
*                        d1 = x(i)-x(nleft)
*                        d2 = x(nright+1)-x(i)
*  # if d1<=d2 with x(nright+1)==x(nright), lowest fixes
*                        if (d1<=d2) break
*  # radius will not decrease by move right
*                        nleft = nleft+1
*                        nright = nright+1
*                        }
*                call lowest(x,y,n,x(i),ys(i),nleft,nright,res,iter>1,rw,ok)
*  # fitted value at x(i)
*                if (!ok) ys(i) = y(i)
*  # all weights zero - copy over value (all rw==0)
*                if (last<i-1) { # skipped points -- interpolate
*                        denom = x(i)-x(last)    # non-zero - proof?
*                        for(j=last+1; j<i; j=j+1){
*                                alpha = (x(j)-x(last))/denom
*                                ys(j) = alpha*ys(i)+(1.0-alpha)*ys(last)
*                                }
*                        }
*                last = i        # last point actually estimated
*                cut = x(last)+delta     # x coord of close points
*                for(i=last+1; i<=n; i=i+1){     # find close points
*                        if (x(i)>cut) break     # i one beyond last pt within cut
*                        if(x(i)==x(last)){      # exact match in x
*                                ys(i) = ys(last)
*                                last = i
*                                }
*                        }
*                i=max0(last+1,i-1)
*  # back 1 point so interpolation within delta, but always go forward
```

```
*                } until(last>=n)
*        do i = 1,n      # residuals
*                res(i) = y(i)-ys(i)
*        if (iter>nsteps) break  # compute robustness weights except last time
*        do i = 1,n
*                rw(i) = abs(res(i))
*        call sort(rw,n)
*        m1 = 1+n/2; m2 = n-m1+1
*        cmad = 3.0*(rw(m1)+rw(m2))      # 6 median abs resid
*        c9 = .999*cmad; c1 = .001*cmad
*        do i = 1,n {
*                r = abs(res(i))
*                if(r<=c1) rw(i)=1.      # near 0, avoid underflow
*                else if(r>c9) rw(i)=0.  # near 1, avoid underflow
*                else rw(i) = (1.0-(r/cmad)**2)**2
*                }
*        }
*  return
*  end
*
*
*
*
*
*                                LOWEST
*
*
*       Calling sequence
*
*       CALL LOWEST(X,Y,N,XS,YS,NLEFT,NRIGHT,W,USERW,RW,OK)
*
*       Purpose
*
*       LOWEST is a support routine for LOWESS and  ordinarily  will
*       not  be  called  by  the  user.   The  fitted  value, YS, is
*       computed  at  the  value,  XS,  of  the   horizontal   axis.
*       Robustness  weights,  RW,  can  be employed in computing the
*       fit.
*
*       Argument description
*
*
*             X = Input; abscissas of the points on the
*                 scatterplot; the values in X must be ordered
*                 from smallest to largest.
*             Y = Input; ordinates of the points on the
*                 scatterplot.
*             N = Input; dimension of X,Y,W, and RW.
*            XS = Input; value of the horizontal axis at which the
*                 smooth is computed.
*            YS = Output; fitted value at XS.
*         NLEFT = Input; index of the first point which should be
*                 considered in computing the fitted value.
*        NRIGHT = Input; index of the last point which should be
*                 considered in computing the fitted value.
*             W = Output; W(I) is the weight for Y(I) used in the
*                 expression for YS, which is the sum from
```

```
*                  I = NLEFT to NRIGHT of W(I)*Y(I); W(I) is
*                  defined only at locations NLEFT to NRIGHT.
*          USERW = Input; logical variable; if USERW is .TRUE., a
*                  robust fit is carried out using the weights in
*                  RW; if USERW is .FALSE., the values in RW are
*                  not used.
*             RW = Input; robustness weights.
*             OK = Output; logical variable; if the weights for the
*                  smooth are all 0.0, the fitted value, YS, is not
*                  computed and OK is set equal to .FALSE.; if the
*                  fitted value is computed OK is set equal to
*
*
*        Method
*
*        The smooth at XS is computed using (robust) locally weighted
*        regression of degree 1.  The tricube weight function is used
*        with h equal to the maximum of XS-X(NLEFT) and X(NRIGHT)-XS.
*        Two  cases  where  the  program  reverts to locally weighted
*        regression of degree 0 are described  in  the  documentation
*        for LOWESS.
*
*
*
*
*  subroutine lowest(x,y,n,xs,ys,nleft,nright,w,userw,rw,ok)
*  real x(n),y(n),w(n),rw(n)
*  logical userw,ok
*  range = x(n)-x(1)
*  h = amax1(xs-x(nleft),x(nright)-xs)
*  h9 = .999*h
*  h1 = .001*h
*  a = 0.0        # sum of weights
*  for(j=nleft; j<=n; j=j+1){     # compute weights (pick up all ties on right)
*         w(j)=0.
*         r = abs(x(j)-xs)
*         if (r<=h9) {    # small enough for non-zero weight
*                 if (r>h1) w(j) = (1.0-(r/h)**3)**3
*                 else      w(j) = 1.
*                 if (userw) w(j) = rw(j)*w(j)
*                 a = a+w(j)
*                 }
*         else if(x(j)>xs)break   # get out at first zero wt on right
*         }
*  nrt=j-1        # rightmost pt (may be greater than nright because of ties)
*  if (a<=0.0) ok = FALSE
*  else { # weighted least squares
*         ok = TRUE
*         do j = nleft,nrt
*                 w(j) = w(j)/a   # make sum of w(j) == 1
*         if (h>0.) {     # use linear fit
*                 a = 0.0
*                 do j = nleft,nrt
*                         a = a+w(j)*x(j) # weighted center of x values
*                 b = xs-a
*                 c = 0.0
*                 do j = nleft,nrt
```

```
*                         c = c+w(j)*(x(j)-a)**2
*                 if(sqrt(c)>.001*range) {
*  # points are spread out enough to compute slope
*                         b = b/c
*                         do j = nleft,nrt
*                                 w(j) = w(j)*(1.0+b*(x(j)-a))
*                         }
*                 }
*         ys = 0.0
*         do j = nleft,nrt
*                 ys = ys+w(j)*y(j)
*         }
*  return
*  end
*
*
*
c  test driver for lowess
c  for expected output, see introduction
      double precision x(20), y(20), ys(20), rw(20), res(20)
      data x /1,2,3,4,5,10*6,8,10,12,14,50/
      data y /18,2,15,6,10,4,16,11,7,3,14,17,20,12,9,13,1,8,5,19/
      call lowess(x,y,20,.25,0,0.,ys,rw,res)
      write(6,*) ys
      call lowess(x,y,20,.25,0,3.,ys,rw,res)
      write(6,*) ys
      call lowess(x,y,20,.25,2,0.,ys,rw,res)
      write(6,*) ys
      end
c**************************************************************
c  Fortran output from ratfor
c
      subroutine lowess(x, y, n, f, nsteps, delta, ys, rw, res)
      integer n, nsteps
      double precision x(n), y(n), f, delta, ys(n), rw(n), res(n)
      integer nright, i, j, iter, last, mid(2), ns, nleft
      double precision cut, cmad, r, d1, d2
      double precision c1, c9, alpha, denom, dabs
      logical ok
      if (n .ge. 2) goto 1
         ys(1) = y(1)
         return
c at least two, at most n points
   1  ns = max(min(int(f*dble(n)), n), 2)
      iter = 1
         goto  3
   2     iter = iter+1
   3     if (iter .gt. nsteps+1) goto  22
c robustness iterations
         nleft = 1
         nright = ns
c index of prev estimated point
         last = 0
c index of current point
         i = 1
   4        if (nright .ge. n) goto  5
c move nleft, nright to right if radius decreases
```

```
               d1 = x(i)-x(nleft)
c if d1<=d2 with x(nright+1)==x(nright), lowest fixes
               d2 = x(nright+1)-x(i)
               if (d1 .le. d2) goto  5
c radius will not decrease by move right
               nleft = nleft+1
               nright = nright+1
               goto  4
c fitted value at x(i)
   5        call lowest(x, y, n, x(i), ys(i), nleft, nright, res, iter
     +     .gt. 1, rw, ok)
            if (.not. ok) ys(i) = y(i)
c all weights zero - copy over value (all rw==0)
            if (last .ge. i-1) goto 9
               denom = x(i)-x(last)
c skipped points -- interpolate
c non-zero - proof?
               j = last+1
                  goto  7
   6              j = j+1
   7              if (j .ge. i) goto  8
                  alpha = (x(j)-x(last))/denom
                  ys(j) = alpha*ys(i)+(1.D0-alpha)*ys(last)
                  goto  6
   8           continue
c last point actually estimated
   9        last = i
c x coord of close points
            cut = x(last)+delta
            i = last+1
               goto  11
  10           i = i+1
  11           if (i .gt. n) goto  13
c find close points
               if (x(i) .gt. cut) goto  13
c i one beyond last pt within cut
               if (x(i) .ne. x(last)) goto 12
                  ys(i) = ys(last)
c exact match in x
                  last = i
  12           continue
               goto  10
c back 1 point so interpolation within delta, but always go forward
  13        i = max(last+1, i-1)
  14        if (last .lt. n) goto  4
c residuals
         do  15 i = 1, n
            res(i) = y(i)-ys(i)
  15        continue
         if (iter .gt. nsteps) goto  22
c compute robustness weights except last time
         do  16 i = 1, n
            rw(i) = dabs(res(i))
  16        continue
         call ssort(rw,n)
         mid(1) = n/2+1
         mid(2) = n-mid(1)+1
```

```
c 6 median abs resid
         cmad = 3.D0*(rw(mid(1))+rw(mid(2)))
         c9 = .999999D0*cmad
         c1 = .000001D0*cmad
         do  21 i = 1, n
            r = dabs(res(i))
            if (r .gt. c1) goto 17
               rw(i) = 1.D0
c near 0, avoid underflow
               goto  20
  17           if (r .le. c9) goto 18
                  rw(i) = 0.D0
c near 1, avoid underflow
                  goto  19
  18              rw(i) = (1.D0-(r/cmad)**2.D0)**2.D0
  19        continue
  20        continue
  21        continue
         goto  2
  22  return
      end


      subroutine lowest(x, y, n, xs, ys, nleft, nright, w, userw
     +, rw, ok)
      integer n
      integer nleft, nright
      double precision x(n), y(n), xs, ys, w(n), rw(n)
      logical userw, ok
      integer nrt, j
      double precision dabs, a, b, c, h, r
      double precision h1, dsqrt, h9, max, range
      range = x(n)-x(1)
      h = max(xs-x(nleft), x(nright)-xs)
      h9 = .999999D0*h
      h1 = .000001D0*h
c sum of weights
      a = 0.D0
      j = nleft
         goto  2
   1     j = j+1
   2     if (j .gt. n) goto  7
c compute weights (pick up all ties on right)
         w(j) = 0.D0
         r = dabs(x(j)-xs)
         if (r .gt. h9) goto 5
            if (r .le. h1) goto 3
               w(j) = (1.D0-(r/h)**3.D0)**3.D0
c small enough for non-zero weight
               goto  4
   3           w(j) = 1.D0
   4        if (userw) w(j) = rw(j)*w(j)
            a = a+w(j)
            goto  6
   5        if (x(j) .gt. xs) goto  7
c get out at first zero wt on right
   6     continue
```

```
         goto  1
c rightmost pt (may be greater than nright because of ties)
   7  nrt = j-1
      if (a .gt. 0.D0) goto 8
         ok = .false.
         goto  16
   8     ok = .true.
c weighted least squares
         do  9 j = nleft, nrt
c make sum of w(j) == 1
            w(j) = w(j)/a
   9        continue
         if (h .le. 0.D0) goto 14
            a = 0.D0
c use linear fit
            do  10 j = nleft, nrt
c weighted center of x values
               a = a+w(j)*x(j)
  10           continue
            b = xs-a
            c = 0.D0
            do  11 j = nleft, nrt
               c = c+w(j)*(x(j)-a)**2
  11           continue
            if (dsqrt(c) .le. .0000001D0*range) goto 13
               b = b/c
c points are spread out enough to compute slope
               do  12 j = nleft, nrt
                  w(j) = w(j)*(b*(x(j)-a)+1.D0)
  12              continue
  13        continue
  14     ys = 0.D0
         do  15 j = nleft, nrt
            ys = ys+w(j)*y(j)
  15        continue
  16  return
      end


      subroutine ssort(a,n)

C Sorting by Hoare method, C.A.C.M. (1961) 321, modified by Singleton
C C.A.C.M. (1969) 185.
           double precision a(n)
           integer iu(16), il(16)
      integer p

      i =1
      j = n
      m = 1
  5   if (i.ge.j) goto 70
c first order a(i),a(j),a((i+j)/2), and use median to split the data
 10   k=i
      ij=(i+j)/2
      t=a(ij)
      if(a(i) .le. t) goto 20
      a(ij)=a(i)
```

```
      a(i)=t
      t=a(ij)
 20   l=j
      if(a(j).ge.t) goto 40
      a(ij)=a(j)
      a(j)=t
      t=a(ij)
      if(a(i).le.t) goto 40
      a(ij)=a(i)
      a(i)=t
      t=a(ij)
      goto 40
 30   a(l)=a(k)
      a(k)=tt
 40   l=l-1
      if(a(l) .gt. t) goto 40
      tt=a(l)
c split the data into a(i to l) .lt. t, a(k to j) .gt. t
 50   k=k+1
      if(a(k) .lt. t) goto 50
      if(k .le. l) goto 30
      p=m
      m=m+1
c split the larger of the segments
      if (l-i .le. j-k) goto 60
      il(p)=i
      iu(p)=l
      i=k
      goto 80
 60   il(p)=k
      iu(p)=j
      j=l
      goto 80
 70   m=m-1
      if(m .eq. 0) return
      i =il(m)
      j=iu(m)
c short sections are sorted by bubble sort
 80   if (j-i .gt. 10) goto 10
      if (i .eq. 1) goto 5
      i=i-1
 90   i=i+1
      if(i .eq. j) goto 70
      t=a(i+1)
      if(a(i) .le. t) goto 90
      k=i
 100  a(k+1)=a(k)
      k=k-1
      if(t .lt. a(k)) goto 100
      a(k+1)=t
      goto 90

      end
```