Savitzky-Golay Smoothing Filters

William H. Press, and Saul A. Teukolsky

Citation: Computers in Physics **4**, 669 (1990); doi: 10.1063/1.4822961
View online: https://doi.org/10.1063/1.4822961
View Table of Contents: https://aip.scitation.org/toc/cip/4/6
Published by the American Institute of Physics

NUMERICAL RECIPES

Savitzky–Golay Smoothing Filters

William H. Press and Saul A. Teukolsky

The notion of a digital filter is most directly applicable to a series of equally spaced data values $f_i \equiv f(t_i)$, where $t_i \equiv t_0 + i\Delta$ for some constant sample spacing Δ and $i = \ldots -2, -1, 0, 1, 2, \ldots$. Then the simplest type of digital filter (the *nonrecursive* or *finite impulse response* filter) replaces each data value f_i by a linear combination g_i of itself and some number of nearby neighbors,

$$g_i = \sum_{n=-n_L}^{n_R} c_n f_{i+n}. \tag{1}$$

Here, n_L is the number of points used "to the left" of a data point i, i.e., earlier than it, while n_R is the number used to the right, i.e., later. A so-called *causal* filter would have $n_R = 0$.

Many whole books[1–3] have been written on digital filter design. Applying a nonrecursive filter is exactly equivalent to multiplying the data's Fourier transform by some filter function. Filter design techniques enable good approximations of certain filter functions (e.g., high-pass, low-pass, bandpass) to be achieved with the smallest number of c_n's.

In this column we discuss a particular type of low-pass filter, termed variously *Savitzky–Golay,*[4] *least squares,*[1] *or DISPO* (Digital Smoothing Polynomial)[5] filters. Rather than having their properties defined in the Fourier domain, and then translated to the time domain of the c_n's, Savitzky–Golay filters derive directly from the time-domain problem of *data smoothing*, and have some highly desirable properties for that application.

The premise of data smoothing is that one is measuring a variable that is both slowly varying and also corrupted by random noise. Then it can sometimes be useful to replace each data point by some kind of local average of surrounding data points. Since nearby points measure very nearly the same underlying value, averaging can reduce the level of noise without (much) biasing the value obtained.

We must comment editorially that the smoothing of data lies in a murky area, beyond the fringe of some better posed, and therefore more highly recommended, techniques that are discussed in the *Numerical Recipes* books.[6–8] If you are fitting data to a parametric model, for example, it is almost always better to use raw data than to use data that has been preprocessed by a smoothing procedure. Another alternative to blind smoothing is so-called "optimal" or Wiener filtering.

William H. Press is a professor of astronomy and physics at Harvard University. Saul A. Teukolsky is a professor of physics and astronomy at Cornell University.

Data smoothing is probably most justified when it is used simply as a graphical technique to guide the eye through a forest of data points all with large error bars; or as a means of making initial *rough* estimates of simple parameters from a graph. In fact, Savitzky–Golay filters were initially (and are still often) used to render visible the relative widths and heights of spectral lines in noisy spectrometric data.

As a starting point for understanding Savitzky–Golay filters, consider the simplest possible averaging procedure: for some fixed $n_L = n_R$, compute each g_i as the average of the data points from f_{i-n_L} to f_{i+n_R}. This is sometimes called *moving window averaging* and corresponds to equation (1) with constant $c_n = 1/(n_L + n_R + 1)$. If the underlying function is constant, or is changing linearly with time (increasing or decreasing), then no bias is introduced into the result. Higher points at one end of the averaging interval are on the average balanced by lower points at the other end. A bias is introduced, however, if the underlying function has a nonzero second derivative. At a local maximum, for example, moving window averaging always reduces the function value. In the spectrometric application, a narrow spectral line has its height reduced and its width increased. Since these parameters are themselves of physical interest, the bias introduced is distinctly undesirable.

Note, however, that moving window averaging does preserve the area under a spectral line, which is its zeroth moment, and also (if the window is symmetric with $n_L = n_R$) its mean position in time, which is its first moment. What is violated is the second moment, equivalent to the line width.

The idea of Savitzky–Golay filtering is to find filter coefficients c_n that preserve higher moments. Equivalently, the idea is to approximate the underlying function within the moving window not by a constant (whose estimate is the average), but by a polynomial of higher order, typically quadratic or quartic: For each point f_i, we least-squares fit a polynomial to all $n_L + n_R + 1$ points in the moving window, and then set g_i to be the value of that polynomial at position i. We make no use of the value of the polynomial at any other point. When we move on to the next point f_{i+1}, we do a whole new least-squares fit using a shifted window.

All these least-squares fits would be laborious if done as described. Luckily, since the process of least-squares fitting involves only a linear matrix inversion, the coefficients of a fitted polynomial are themselves linear in the values of the data. That means that we can do all the fitting in advance, for fictitious data consisting of all zeros

Box 1.

```
      SUBROUTINE savgol(c,np,nl,nr,ld,m)
      INTEGER ld,m,nl,np,nr,MMAX
      REAL c(np)
      PARAMETER (MMAX=6)
C     USES lubksb,ludcmp
        Returns in c(1:np), in wraparound order (N.B.!) consistent with the argument respns in
        routine convlv, a set of Savitzky-Golay filter coefficients. nl is the number of leftward (past)
        data points used, while nr is the number of rightward (future) data points, making the total
        number of data points used nl+nr+1. ld is the order of the derivative desired (e.g., ld= 0
        for smoothed function). m is the order of the smoothing polynomial, also equal to the highest
        conserved moment; usual values are m= 2 or m= 4.
      INTEGER imj,ipj,j,k,kk,mm,indx(MMAX+1)
      REAL d,fac,sum,a(MMAX+1,MMAX+1),b(MMAX+1)
      if(np.lt.nl+nr+1.or.nl.lt.0.or.nr.lt.0.or.ld.gt.m.or.m.gt.MMAX
     *     .or.nl+nr.lt.m) pause 'bad args in savgol'
      do 14 ipj=0,2*m                    Set up the normal equations of the desired least-squares fit.
        sum=0.
        if(ipj.eq.0)sum=1.
        do 11 k=1,nr
          sum=sum+float(k)**ipj
        enddo 11
        do 12 k=1,nl
          sum=sum+float(-k)**ipj
        enddo 12
        mm=min(ipj,2*m-ipj)
        do 13 imj=-mm,mm,2
          a(1+(ipj+imj)/2,1+(ipj-imj)/2)=sum
        enddo 13
      enddo 14
      call ludcmp(a,m+1,MMAX+1,indx,d)   Solve them: LU decomposition.
      do 15 j=1,m+1
        b(j)=0.
      enddo 15
      b(ld+1)=1.          Right-hand side vector is unit vector, depending on which derivative we want.
      call lubksb(a,m+1,MMAX+1,indx,b)   Backsubstitute, giving one row of the inverse matrix.
      do 16 kk=1,np                      Zero the output array (it may be bigger than number of co-
        c(kk)=0.                         efficients).
      enddo 16
      do 18 k=-nl,nr                     Each Savitzky-Golay coefficient is the dot product of powers
        sum=b(1)                         of an integer with the inverse matrix row.
        fac=1.
        do 17 mm=1,m
          fac=fac*k
          sum=sum+b(mm+1)*fac
        enddo 17
        kk=mod(np-k,np)+1                Store in wraparound order.
        c(kk)=sum
      enddo 18
      return
      END
```

except for a single 1, and then do the fits on the real data just by taking linear combinations. This is the key point, then: There are particular sets of filter coefficients c_n for which equation (1) "automatically" accomplishes the process of polynomial least-squares fitting inside a moving window.

To derive such coefficients, consider how g_0 might be obtained: We want to fit a polynomial of degree M in i, namely, $a_0 + a_1 i + \cdots + a_M i^M$ to the values $f_{-n_L}, \ldots, f_{n_R}$. Then g_0 will be the value of that polynomial at $i = 0$, namely, a_0. The design matrix for this problem (see *Numerical Recipes*,[6-8] Sec. 14.3) is

$$A_{ij} = i^j, \quad i = -n_L, \ldots, n_R, \quad j = 0, \ldots, M, \tag{2}$$

and the normal equations for the vector of a_j's in terms of the vector of f_i's is in matrix notation

$$(\mathbf{A}^T \cdot \mathbf{A}) \cdot \mathbf{a} = \mathbf{A}^T \cdot \mathbf{f}$$

or

$$\mathbf{a} = (\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot (\mathbf{A}^T \cdot \mathbf{f}). \tag{3}$$

We also have the specific forms

$$\{\mathbf{A}^T \cdot \mathbf{A}\}_{ij} = \sum_{k=-n_L}^{n_R} A_{ki} A_{kj} = \sum_{k=-n_L}^{n_R} k^{i+j} \tag{4}$$

and

$$\{\mathbf{A}^T \cdot \mathbf{f}\}_j = \sum_{k=-n_L}^{n_R} A_{kj} f_k = \sum_{k=-n_L}^{n_R} k^j f_k. \tag{5}$$

Since the coefficient c_n is the component a_0 when $\mathbf{f}$ is replaced by the unit vector $\mathbf{e}_n$, $-n_L \leqslant n_R$, we have

$$c_n = \{(\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot (\mathbf{A}^T \cdot \mathbf{e}_n)\}_0 = \sum_{m=0}^{M} \{(\mathbf{A}^T \cdot \mathbf{A})^{-1}\}_{0m} n^m. \tag{6}$$

Note that equation (6) says that we need only one row of the inverse matrix. Numerically we can get this by LU decomposition with only a single backsubstitution. Note also that the size of the matrix is set by the order M, and not by the size of the filter $(n_L + n_R + 1)$.

The subroutine savgol, shown in Box 1, implements equation (6). As input, it takes the parameters nl $= n_L$, nr $= n_R$, and m $= M$ (the desired order). Also input is np, the physical length of the output array c, and a parameter ld which for the purposes of this column should always be set to zero. (In fact, ld specifies which coefficient among the a_i's should be returned. We are here always interested in a_0. For another purpose, namely the computation of numerical derivatives, to be discussed in a future column, the value ld $= 1$ becomes interesting.)

There is almost never a good reason to use large M values, say >4. For $M \leqslant 4$, the matrix is never ill-conditioned. For $M > 4$, when also $n_L \gg n_R$ or $n_L \ll n_R$, it sometimes is ill-conditioned; the *Numerical Recipes* routine mprove can then be used to improve the solution.

As output, savgol returns the coefficients c_n, for $-n_L \leqslant n \leqslant n_R$. These are stored in c in "wraparound order"; that is, c_0 is in c(1), c_{-1} is in c(2), and so on for further negative indices. The value c_1 is stored in c(np), c_2 in c(np $-$ 1), and so on for positive indices. This order may seem arcane, but it is the natural one where causal filters have nonzero coefficients in low array elements of c. It is also the order required by the *Numerical Recipes* subroutine convlv, which uses fast Fourier transform techniques to apply a digital filter to a data set.

Table I shows some typical output from savgol. For orders 2 and 4, the coefficients of Savitzky–Golay filters with several choices of n_L and n_R are shown. The central column is the coefficient applied to the data f_i in obtaining the smoothed g_i. Coefficients to the left are applied to earlier data; to the right, to later. The coefficients always add (within roundoff error) to unity. One sees that, as befits a smoothing operator, the coefficients always have a central positive lobe, but with smaller, outlying corrections of both positive and negative sign. In practice, the Savitzky–Golay filters are most useful for much larger values of n_L and n_R, since these few-point formulas can accomplish only a relatively small amount of smoothing.

TABLE I. Typical output from savgol.

Order	n_L	n_R	Sample Savitzky–Golay coefficients										
2	2	2				−0.086	0.343	0.486	0.343	−0.086			
2	3	1			−0.143	0.171	0.343	0.371	0.257				
2	4	0		0.086	−0.143	−0.086	0.257	0.886					
2	5	5	−0.084	0.021	0.103	0.161	0.196	0.207	0.196	0.161	0.103	0.021	−0.084
4	4	4		0.035	−0.128	0.070	0.315	0.417	0.315	0.070	−0.128	0.035	
4	5	5	0.042	−0.105	−0.023	0.140	0.280	0.333	0.280	0.140	−0.023	−0.105	0.042

Figure 1 shows a numerical experiment using a 33-point smoothing filter, that is, $n_L = n_R = 16$. The upper panel shows a test function, constructed to have six "bumps" of varying widths, all of height eight units. To this function Gaussian white noise of unit variance has been added. (The test function without noise is shown as the dotted curves in the center and lower panels.) The widths of the bumps (full width at half of maximum, or FWHM) are 140, 43, 24, 17, 13, and 10, respectively.

The middle panel of Fig. 1 shows the result of smoothing by a moving window average. One sees that the window of width 33 does quite a nice job of smoothing the broadest bump, but that the narrrower bumps suffer considerable loss of height and increase of width. The underlying signal (dotted) is very badly represented.

The lower panel shows the result of smoothing with a Savitzky–Golay filter of the identical width, and degree $M = 4$. One sees that the heights and widths of the bumps are quite extraordinarily preserved. A trade-off is that the broadest bump is less smoothed. That is because the central positive lobe of the Savitzky–Golay filter coefficients fills only a fraction of the full 33-point width. As a rough guideline, best results are obtained when the full width of the degree 4 Savitzky–Golay filter is between 1 and 2 times the FWHM of desired features in the data. (References 5 and 9 give additional practical hints.)

Figure 2 show the result of smoothing the same noisy "data" with broader Savitzky–Golay filters of three different orders. Here, we have $n_L = n_R = 32$ (65-point filter) and $M = 2,4,6$. One sees that, when the bumps are

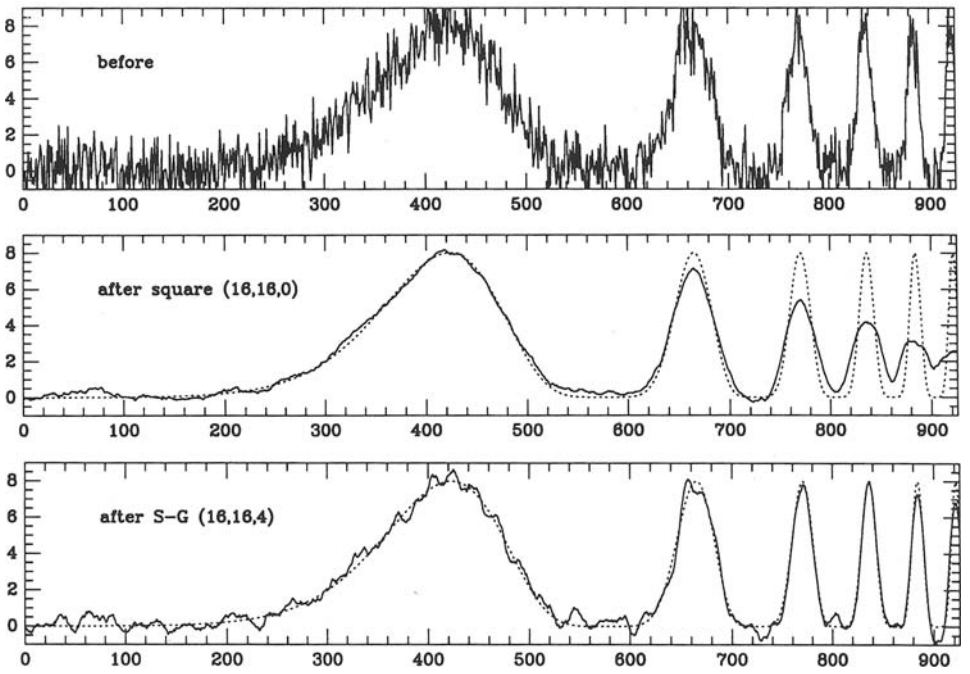

FIG. 1(a) Synthetic noisy data consisting of a sequence of progressively narrower bumps, and additive Gaussian white noise. (b) Result of smoothing the data by a simple moving window average. The window extends 16 points leftward and rightward, for a total of 33 points. Note that narrow features are broadened and suffer corresponding loss of amplitude. The dotted curve is the underlying function used to generate the synthetic data. (c) Result of smoothing the data by a Savitzky-Golay smoothing filter (of degree 4) using the same 33 points. While there is less smoothing of the broadest feature, narrower features have their heights and widths preserved.

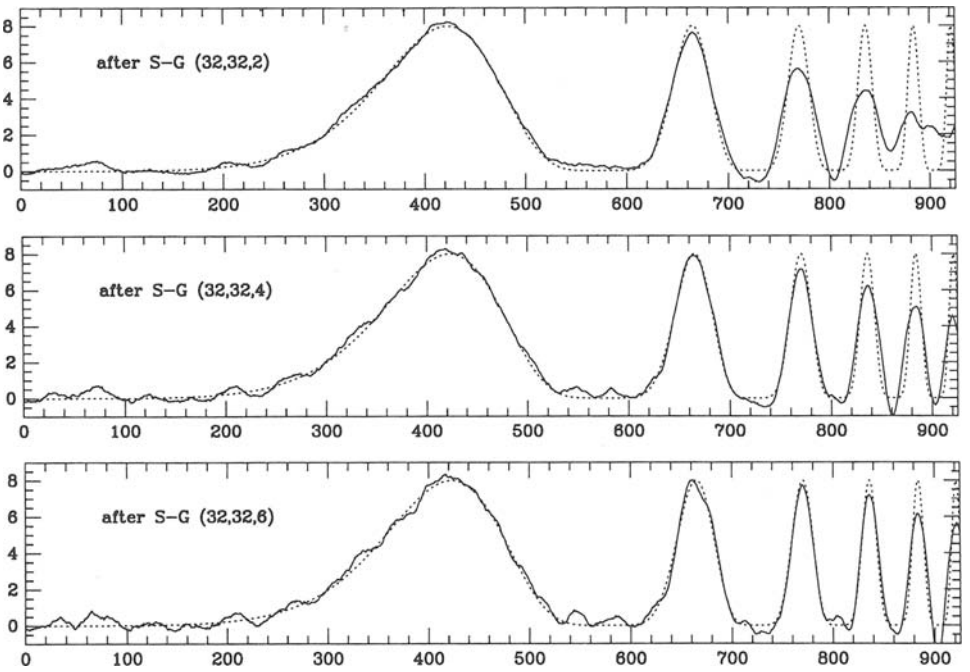

FIG. 2. Result of applying wider 65-point Savitzky-Golay filters to the same data set as in Fig. 1. Top: degree 2. Center: degree 4. Bottom: degree 6. All of these filters are inoptimally broad for the resolution of the narrow features. Higher-order filters do best at preserving feature heights and widths, but do less smoothing on broader features.

too narrow with respect to the filter size, then even the Savitzky–Golay filter must at some point give out. The higher-order filter manages to track narrower features, but at the cost of less smoothing on broad features.

To summarize: Within limits, Savitzky–Golay filtering does manage to provide smoothing without loss of resolution. It does this by assuming that relatively distant data points have a usable redundancy that can be used to reduce the level of noise. The specific nature of this assumed redundancy is that the underlying function should be locally well fitted by a polynomial. When this is true, as it is for smooth line profiles not too much narrower than the filter width, then the performance of Savitzky–Golay filters can be spectacular. When it is not true, then these filters have no compelling advantage over other classes of smoothing filter coefficients.

A final remark concerns irregularly sampled data, where the values f_i are not uniformly spaced in time. The obvious generalization of Savitzky–Golay filtering would be to do a least-squares fit within a moving window around each data point, one containing a fixed number of data points to the left (n_L) and right (n_R). Because of the irregular spacing, however, there is no way to obtain universal filter coefficients applicable to more than one data point. One must instead do the actual least-squares fits for each data point. This becomes computationally burdensome for larger n_L, n_R, and M.

As a cheap alternative, one can simply pretend that the data points *are* equally spaced. This amounts to virtually shifting, within each moving window, the data points to equally spaced positions. Such a shift introduces the equivalent of an additional source of noise into the function values. In those cases where smoothing is useful, this noise will often be much smaller than the noise already present. Specifically, if the location of the points is approximately random within the window, then a rough criterion is this: If the change in f across the full width of the $N = n_L + n_R + 1$ point window is less than $\sqrt{N/2}$ times the measurement noise on a single point, then the cheap method can be used. ■

In our next column: Methods for computing numerical derivatives.

References

1. R. W. Hamming, *Digital Filters* (Prentice-Hall, Englewood Cliffs, NJ, 1983), 2nd ed.
2. A. Antoniou, *Digital Filters: Analysis and Design* (McGraw-Hill, New York, 1979).
3. T. W. Parks and C. S. Burrus, *Digital Filter Design* (Wiley, New York, 1987).
4. A. Savitzky and M. J. E. Golay, Anal. Chem. **36**, 1627 (1964).
5. H. Ziegler, Appl. Spectrosc. **35**, 88 (1981).
6. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing* (Cambridge U.P., New York, 1986).
7. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing* (Cambridge U.P., New York, 1988).
8. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in Pascal: The Art of Scientific Computing* (Cambridge U. P., New York, 1989).
9. M. U. A. Bromba and H. Ziegler, Anal. Chem. **53**, 1583 (1981).