An Introduction to Signal Processing in Chemical Analysis

An illustrated essay with software available for free download

Last updated June, 2009

PDF format: http://terpconnect.umd.edu/~toh/spectrum/IntroToSignalProcessing.pdf

Web format: http://terpconnect.umd.edu/~toh/spectrum/TOC.html

Tom O'Haver
Professor Emeritus
Department of Chemistry and Biochemistry
University of Maryland at College Park
E-mail: toh@umd.edu

Foreword

The interfacing of analytical instrumentation to small computers for the purpose of on-line data acquisition has now become almost standard practice in the modern chemistry laboratory. Using widely-available, low-cost microcomputers and off-the-shelf add-in components, it is now easier than ever to acquire data quickly in digital form. In what ways is on-line digital data acquisition superior to the old methods such as the chart recorder? Some of the advantages are obvious, such as archival storage and retrieval of data and post-run re-plotting with adjustable scale expansion. Even more important, however, there is the possibility of performing post-run data analysis and signal processing. There are a large number of computer-based numerical methods that can be used to reduce noise, improve the resolution of overlapping peaks, compensate for instrumental artifacts, test hypotheses, optimize measurement strategies, diagnose measurement difficulties, and decompose complex signals into their component parts. These techniques can often make difficult measurements easier by extracting more information from the available data. Many of these techniques are based on laborious mathematical procedures that were not practical before the advent of computerized instrumentation. It is important for chemistry students to appreciate the capabilities and the limitations of these modern signal processing techniques.

In the chemistry curriculum, signal processing may be covered as part of a course on instrumental analysis (1, 2), electronics for chemists (3), laboratory interfacing (4), or basic chemometrics (5). The purpose of this paper is to give a general introduction to some of the most widely used signal processing techniques and to give illustrations of their applications in analytical chemistry. This essay covers only elementary topics and is limited to only basic mathematics. For more advanced topics and for a more rigorous treatment of the underlying mathematics, refer to the extensive literature on chemometrics.

This tutorial makes use of a freeware signal-processing program called SPECTRUM that was used to produce many of the illustrations. Additional examples were developed in Matlab, a high-performance commercial numerical computing environment and programming language that is widely used in research. Paragraphs in gray at the end of each section in this essay describe the related capabilities of each of these programs.

Signal arithmetic

The most basic signal processing functions are those that involve simple signal arithmetic: point-by-point addition, subtraction, multiplication, or division of two signals or of one signal and a constant. Despite their mathematical simplicity, these functions can be very useful. For example, in the left part of Figure 1 (Window 1) the top curve is the absorption spectrum of an extract of a sample of oil shale, a kind of rock that is is a source of petroleum.

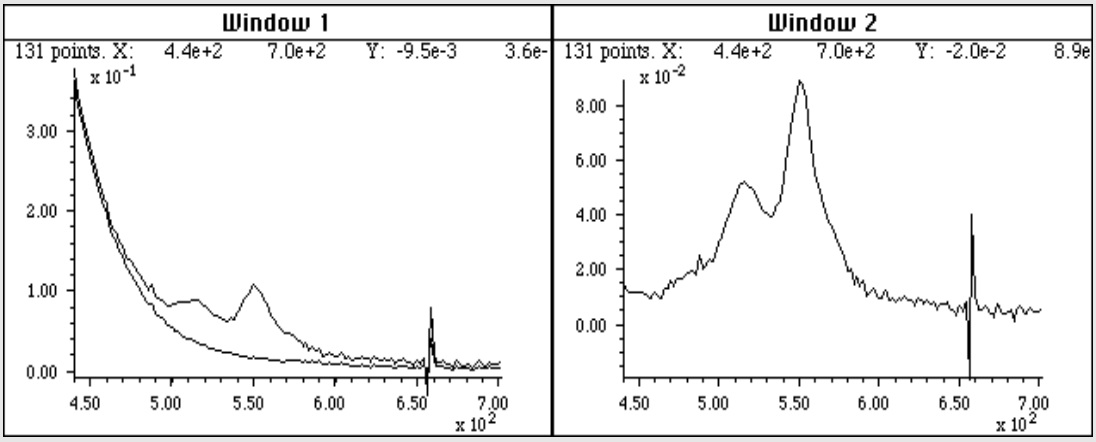


Figure 1. *A simple point-by--point subtraction of two signals allows the background (bottom curve on the left) to be subtracted from a complex sample (top curve on the left), resulting in a clearer picture of what is really in the sample (right).*

This spectrum exhibits two absorption bands, at about 515 nm and 550 nm, that are due to a class of molecular fossils of chlorophyll called *porphyrins.* (Porphyrins are used as geomarkers in oil exploration). These bands are superimposed on a background absorption caused by the extracting solvents and by non-porphyrin compounds extracted from the shale. The bottom curve is the spectrum of an extract of a non-porphyrin-bearing shale, showing only the background absorption. To obtain the spectrum of the shale extract without the background, the background (bottom curve) is simply subtracted from the sample spectrum (top curve). The difference is shown in the right in Window 2 (note the change in Y-axis scale). In this case the removal of the background is not perfect, because the background spectrum is measured on a separate shale sample. However, it works well enough that the two bands are now seen more clearly and it is easier to measure precisely their absorbances and wavelengths.

In this example and the one below, the assumption is being made that the two signals in Window 1 have the same x-axis values, that is, that both spectra are digitized at the same set of wavelengths. Strictly speaking this operation would not be valid if two spectra were digitized over different wavelength ranges or with different intervals between adjacent points. The x-axis values much match up point for point. In practice, this is very often the case with data sets acquired within one experiment on one instrument, but the experimenter must take care if the instruments settings are changed or if data from two experiments or two instrument are combined. (Note: It is possible to use the mathematical technique of interpolation to change the number of points or the x-axis intervals of

signals; the results are only approximate but often close enough in practice).

Sometimes one needs to know whether two signals have the same shape, for example in comparing the spectrum of an unknown to a stored reference spectrum. Most likely the concentrations of the unknown and reference, and therefore the amplitudes of the spectra, will be different. Therefore a direct overlay or subtraction of the two spectra will not be useful. One possibility is to compute the point-by-point ratio of the two signals; if they have the same shape, the ratio will be a constant. For example, examine Figure 2.

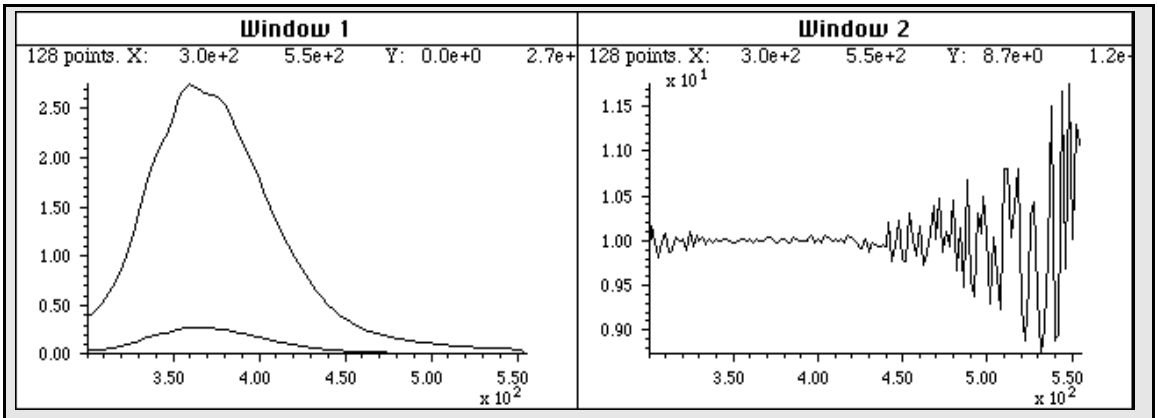


Figure 2. *Do the two spectra on the left have the same shape? They certainly do not look the same, but that may simply be due to that fact that one is much weaker that the other. The ratio of the two spectra, shown in the right part (Window 2), is relatively constant from 300 to 440 nm, with a value of 10 +/- 0.2. This means that the shape of these two signals is very nearly identical over this wavelength range.*

The left part (Window 1) shows two superimposed spectra, one of which is much weaker than the other. But do they have the same shape? The ratio of the two spectra, shown in the right part (Window 2), is relatively constant from 300 to 440 nm, with a value of 10 +/- 0.2. This means that the shape of these two signals is the same, within about +/-2 %, over this wavelength range, and that top curve is about 10 times more intense than the bottom one. Above 440 nm the ratio is not even approximately constant; this is caused by *noise,* which is the topic of the next section.

Simple signal arithmetic operations such as these are easily done in a spreadsheet, any general-purpose programming language, or a dedicated signal-processing program such as SPECTRUM, which is available for free download.

SPECTRUM includes addition and multiplication of a signal with a constant; addition, subtraction, multiplication, and division of two signals; normalization, and a large number of other basic math functions (log, ln, antilog, square root, reciprocal, etc).

In Matlab, math operations on signals are especially powerful because the variables in Matlab can be either *scalar* (single values), *vector* (like a row or a column in a spreadsheet), representing one entire signal, spectrum or chromatogram, or *matrix* (like a rectangular block of cells in a spreadsheet), representing a *set* of signals. For example, in Matlab you could define two vectors **`a=[1 2 5 2 1]`** and **`b=[4 3 2 1 0]`**. Then to subtract B from A you would just type **`a-b`**, which gives the result **`[-3 -1 3 1 1]`**. To multiply A times B point by point, you would type **`a.*b`**, which gives the result **`[4 6 10 2 0]`**. If you have an entire spectrum in the variable **`a`**, you can plot it just by

typing **`plot(a)`**. And if you also had a vector `w` of x-axis values (such as wavelengths), you can plot **`a`** vs **`w`** by typing **`plot(w,a)`**. The subtraction of two spectra **`a`** and **`b`**, as in Figure 1, can be performed simply by writing **`a-b`**. To plot the difference, you would write **`plot(a-b)`**. Likewise, to plot the ratio of two spectra, as in Figure 2, you would write **`plot(a./b)`**. Moreover, Matlab is a programming language that can automate complex sequences of operations by saving them in scripts and functions.

Signals and noise

Experimental measurements are never perfect, even with sophisticated modern instruments. Two main types or measurement errors are recognized: *systematic error,* in which every measurement is either less than or greater than the "correct" value by a fixed percentage or amount, and *random error,* in which there are unpredictable variations in the measured signal from moment to moment or from measurement to measurement. This latter type of error is often called *noise*, by analogy to acoustic noise. There are many sources of noise in physical measurements, such as building vibrations, air currents, electric power fluctuations, stray radiation from nearby electrical apparatus, interference from radio and TV transmissions, random thermal motion of molecules, and even the basic quantum nature of matter and energy itself.

In spectroscopy, three fundamental type of noise are recognized: photon noise, detector noise, and flicker (fluctuation) noise. Photon noise (often the limiting noise in instruments that use photomultiplier detectors), is proportional to the square root of light intensity, and therefore the SNR is proportional to the square root of light intensity and directly proportional to the slit width. Detector noise (often the limiting noise in instruments that use solid-state photodiode detectors) is independent of the light intensity and therefore the detector SNR is directly proportional to the light intensity and to the square of the monochromator slit width. Flicker noise, caused by light source instability, vibration, sample cell positioning errors, sample turbulence, light scattering by suspended particles, dust, bubbles, etc., is directly proportional to the light intensity, so the flicker SNR is not decreased by increasing the slit width. Flicker noise can usually be reduced or eliminated by using specialized instrument designs such as double-beam, dual wavelength, diode array, and wavelength modulation.

The quality of a signal is often expressed quantitatively as the *signal-to-noise ratio* (SNR) which is the ratio of the true signal amplitude (e.g. the average amplitude or the peak height) to the standard deviation of the noise. Signal-to-noise ratio is inversely proportional to the relative standard deviation of the signal amplitude. Measuring the signal-to-noise ratio usually requires that the noise be measured separately, in the absence of signal. Depending on the type of experiment, it may be possible to acquire readings of the noise alone, for example on a segment of the baseline before or after the occurrence of the signal. However, if the magnitude of the noise depends on the level of the signal (as in photon noise or flicker noise in spectroscopy), then the experimenter must try to produce a constant signal level to allows measurement of the noise on the signal. In a few cases, where it is possible to model the shape of the signal exactly by means of a mathematical function, the noise may be estimated by subtracting the model signal from the experimental signal.

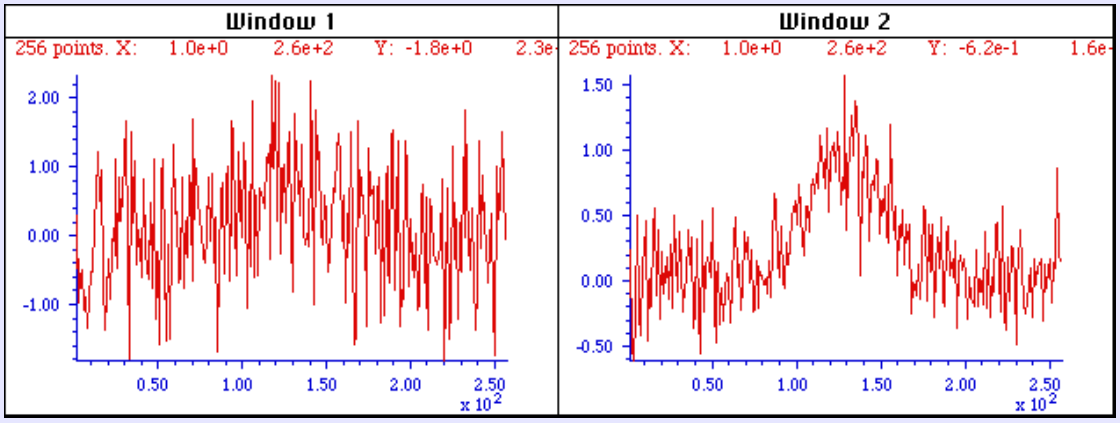


Figure 3. *Window 1 (left) is a single measurement of a very noisy signal. There is actually a broad peak near the center of this signal, but it is not possible to measure its position, width, and height accurately because the signal-to-noise ratio is very poor (less than 1). Window 2 (right) is the average of 9 repeated measurements of this signal, clearly showing the peak emerging from the noise. The expected improvement in signal-to-noise ratio is 3 (the square root of 9). Often it is possible to average hundreds of measurement, resulting is much more substantial improvement.*

One of the fundamental problems in signal measurement is distinguishing the noise from the signal. Sometimes the two can be partly distinguished on the basis of frequency components: for example, the signal may contain mostly low-frequency components and the noise may be located a higher frequencies. This is the basis of filtering and smoothing. But the thing that really distinguishes signal from noise is that random noise is not the same from one measurement of the signal to the next, whereas the genuine signal is at least partially reproducible. So if the signal can be measured more than once, use can be made of this fact by measuring the signal over and over again as fast as practical and adding up all the measurements point-by-point. This is called *ensemble averaging*, and it is one of the most powerful methods for improving signals, when it can be applied. For this to work properly, the noise must be random and the signal must occur at the same time in each repeat. An example is shown in Figure 3. 3.

SPECTRUM includes several functions for measuring signals and noise, plus a signal-generator that can be used to generate artificial signals with Gaussian and Lorentzian bands, sine waves, and normally-distributed random noise. Matlab has built-in functions that can be used for measuring and plotting signals and noise, such as mean, max, min, range, std, plot, hist. You can also create user-defined functions to automate commonly-used algorithms. Some examples that you can download and use are these user-defined functions to calculate typical peak shapes commonly encountered in analytical chemistry, gaussian and lorentzian, and typical types of random noise (whitenoise, pinknoise), which can be useful in modeling and simulating analytical signals and testing measurement techniques. (If you are viewing this document on-line, you can Ctrl-click on these links to inspect the code). Once you have created or downloaded those functions, you can use them to plot a simulated noisy peak such as in Figure 3 by typing
`x=[1:256];plot(x,gaussian(x,128,64)+whitenoise(x))`.

Smoothing

In many experiments in physical science, the true signal amplitudes (y-axis values) change rather smoothly as a function of the x-axis values, whereas many kinds of noise are seen as rapid, random changes in amplitude from point to point within the signal. In the latter situation it is common practice to attempt to reduce the noise by a process called *smoothing*. In smoothing, the data points of a signal are modified so that individual points that are higher than the immediately adjacent points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased. This naturally leads to a smoother signal. As long as the true underlying signal is actually smooth, then the true signal will not be much distorted by smoothing, but the noise will be reduced.

Smoothing algorithms. The simplest smoothing algorithm is the *rectangular* or *unweighted sliding-average smooth*; it simply replaces each point in the signal with the average of *m* adjacent points, where *m* is a positive integer called the *smooth width*. For example, for a 3-point smooth (*m*=3):

$$S_j = \frac{Y_{j-1} + Y_j + Y_{j+1}}{3}$$

for j = 2 to n-1, where S_j the j^{th} point in the smoothed signal, Y_j the j^{th} point in the original signal, and n is the total number of points in the signal. Similar smooth operations can be constructed for any desired smooth width, *m*. Usually *m* is an odd number. If the noise in the data is "white noise" (that is, evenly distributed over all frequencies) and its standard deviation is *s*, then the standard deviation of the noise remaining in the signal after the first pass of an unweighted sliding-average smooth will be approximately s over the square root of *m* (*s*/sqrt(*m*)), where *m* is the smooth width.

The *triangular smooth* is like the rectangular smooth, above, except that it implements a weighted smoothing function. For a 5-point smooth (*m*=5):

$$S_j = \frac{Y_{j-2} + 2Y_{j-1} + 3Y_j + 2Y_{j+1} + Y_{j+2}}{9}$$

for j = 3 to n-2, and similarly for other smooth widths. This is equivalent to two passes of a 3-point rectangular smooth. This smooth is more effective at reducing high-frequency noise in the signal than the simpler rectangular smooth. Note that again in this case, the width of the smooth *m* is an odd integer and the smooth coefficients are symmetrically balanced around the central point, which is important point because it preserves the x-axis position of peaks and other features in the signal. (This is especially critical for analytical and spectroscopic applications because the peak positions are sometimes important measurement objectives).

Note that we are assuming here that the x-axis intervals of the signal is uniform, that is, that the difference between the x-axis values of adjacent points is the same throughout the signal. This is also assumed in many of the other signal-processing techniques described in this essay, and it is a very common (but not necessary) characteristic of signals that are acquired by automated and computerized equipment.

Noise reduction. Smoothing usually reduces the noise in a signal. If the noise is "white" (that is, evenly distributed over all frequencies) and its standard deviation is *s*, then the

standard deviation of the noise remaining in the signal after one pass of a triangular smooth will be approximately *s**0.8/sqrt(*m*), where *m* is the smooth width. Smoothing operations can be applied more than once: that is, a previously-smoothed signal can be smoothed again. In some cases this can be useful if there is a great deal of high-frequency noise in the signal. However, the noise reduction for white noise is less less in each successive smooth. For example, three passes of a rectangular smooth reduces white noise by a factor of approximately *s**0.7/sqrt(*m*), only a slight improvement over two passes (triangular smooth).

Edge effects and the lost points problem. Note in the equations above that the 3-point rectangular smooth is defined only for j = 2 to n-1. There is not enough data in the signal to define a complete 3-point smooth for the first point in the signal (j = 1) or for the last point (j = n) , because there are no data points before the first point or after the last point. Similarly, a 5-point smooth is defined only for j = 3 to n-2, and therefore a smooth can not be calculated for the first two points or for the last two points. In general, for an *m*-width smooth, there will be (*m*-1)/2 points at the beginning of the signal and (*m*-1)/2 points at the end of the signal for which a complete *m*-width smooth can not be calculated. What to do? There are two approaches. One is to accept the loss of points and trim off those points or replace them with zeros in the smooth signal. (That's the approach taken in the figures in this paper). The other approach is to use progressively smaller smooths at the ends of the signal, for example to use 2, 3, 5, 7... point smooths for signal points 1, 2, 3,and 4..., and for points n, n-1, n-2, n-3..., respectively. The later approach may be preferable if the edges of the signal contain critical information, but it increases execution time.

Examples of smoothing. A simple example of smoothing is shown in Figure 4. The left half of this signal is a noisy peak. The right half is the same peak after undergoing a triangular smoothing algorithm. The noise is greatly reduced while the peak itself is hardly changed. Smoothing increases the signal-to-noise ratio and allows the signal characteristics (peak position, height, width, area, etc.) to be measured more accurately, especially when computer-automated methods of locating and measuring peaks are being employed.

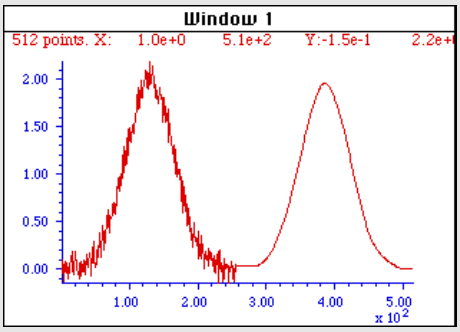
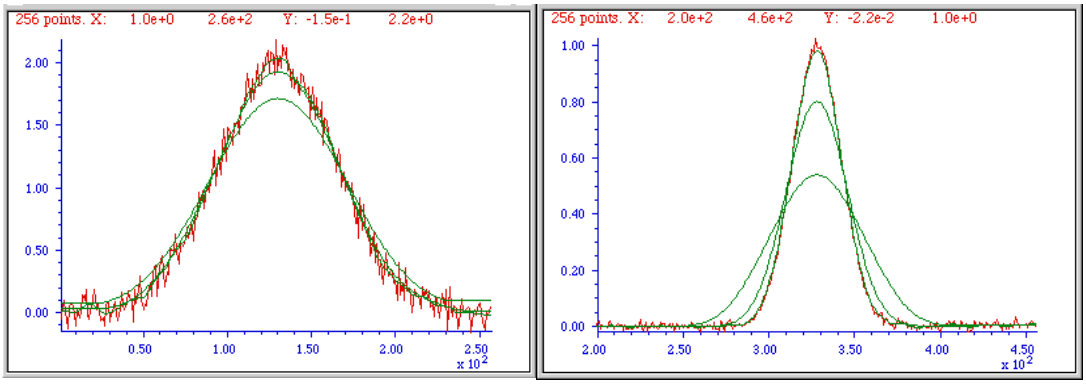


Figure 4. *The left half of this signal is a noisy peak. The right half is the same peak after undergoing a* ***smoothing*** *algorithm. The noise is greatly reduced while the peak itself is hardly changed, making it easier to measure the peak position, height, and width.*

The larger the smooth width, the greater the noise reduction, but also the greater the possibility that the signal will be distorted by the smoothing operation. The optimum choice of smooth width depends upon the width and shape of the signal and the digiti-

zation interval. For peak-type signals, the critical factor is the *smoothing ratio*, the ratio between the smooth width *m* and the number of points in the half-width of the peak. In general, increasing the smoothing ratio improves the signal-to-noise ratio but causes a reduction in amplitude and in increase in the bandwidth of the peak.



The figures above show examples of the effect of three different smooth widths on noisy gaussian-shaped peaks. In the figure on the left, the peak has a (true) height of 2.0 and there are 80 points in the half-width of the peak. The red line is the original unsmoothed peak. The three superimposed green lines are the results of smoothing this peak with a triangular smooth of width (from top to bottom) 7, 25, and 51 points. Because the peak width is 80 points, the *smooth ratios* of these three smooths are 7/80 = 0.09, 25/80 = 0.31, and 51/80 = 0.64, respectively. As the smooth width increases, the noise is progressively reduced but the peak height also is reduced slightly. For the largest smooth, the peak width is slightly increased. In the figure on the right, the original peak (in red) has a true height of 1.0 and a half-width of 33 points. (It is also less noisy than the example on the left.) The three superimposed green lines are the results of the same three triangular smooths of width (from top to bottom) 7, 25, and 51 points. But because the peak width in this case is only 33 points, the *smooth ratios* of these three smooths are larger: 0.21, 0.76, and 1.55, respectively. You can see that the peak distortion effect (reduction of peak height and increase in peak width) is greater for the narrower peak because the smooth ratios are higher. Smooth ratios of greater than 1.0 are seldom used because of excessive peak distortion. Note that even in the worst case, the peak positions are not effected (assuming that the original peaks were symmetrical and not overlapped by other peaks).

It's important to point out that smoothing results such as illustrated in the figure above may be deceptively impressive because they employ a *single sample* of a noisy signal that is smoothed to different degrees. This causes the viewer to *underestimate the contribution of low-frequency noise*, which is hard to estimate visually because there are so few low-frequency cycles in the signal record. This error can be remedied by taking a large number of independent samples of noisy signal. This is illustrated in the Interactive Smoothing module for Matlab, which includes a "Resample" control that swaps the noise in the signal with different random noise samples, to demonstrate the low-frequency noise that remains in the signal after smoothing. This gives a much more realistic impression of the performance of smoothing

Optimization of smoothing. Which is the best smooth ratio? It depends on the purpose of the peak measurement. If the objective of the measurement is to measure the true peak height and width, then smooth ratios below 0.2 should be used. (In the example on the

left, the original peak (red line) has a peak height greater than the true value 2.0 because of the noise, whereas the smoothed peak with a smooth ratio of 0.09 has a peak height that is much closer to the correct value). But if the objective of the measurement is to measure the peak position (x-axis value of the peak), much larger smooth ratios can be employed if desired, because smoothing has no effect at all on the peak position of a symmetrical peak (unless the increase in peak width is so much that it causes adjacent peaks to overlap).

In quantitative analysis applications, the peak height reduction caused by smoothing is not so important, because in most cases calibration is based on the signals of standard solutions. If the same signal processing operations are applied to the samples and to the standards, the peak height reduction of the standard signals will be exactly the same as that of the sample signals and the effect will cancel out exactly. In such cases smooth widths from 0.5 to 1.0 can be used if necessary to further improve the signal-to-noise ratio. (The noise is reduced by approximately the square root of the smooth width). In practical analytical chemistry, absolute peak height measurements are seldom required; calibration against standard solutions is the rule. (Remember: the objective of a quantitative analytical spectrophotometric procedure is not to measure absorbance but rather to measure the concentration of the analyte.) It is very important, however, to apply *exactly* the same signal processing steps to the standard signals as to the sample signals, otherwise a large systematic error may result.

When should you smooth a signal? There are two reasons to smooth a signal: (1) for cosmetic reasons, to prepare a nicer-looking graphic of a signal for visual inspection or publication, and (2) if the signal will be subsequently processed by an algorithm that would be adversely effected by the presence of too much high-frequency noise in the signal, for example if the location of maxima, mimima, or inflection points in the signal is to be automatically determined by detecting zero-crossings in derivatives of the signal. But one common situation where you should ***not*** smooth signals is prior to least-squares curve fitting, because all smoothing algorithms are at least slightly "lossy", entailing at least some change in signal shape and amplitude. If these requirements conflict, care must be used in the design of algorithms. For example, in a popular technique for peak finding and measurement, peaks are located by detecting downward zero-crossings in the smoothed first derivative, but the position, height, and width of each peak is determined by least-squares curve-fitting of a segment of original unsmoothed data in the vicinity of the zero-crossing. Thus, even if heavy smoothing is necessary to provide reliable discrimination against noise peaks, the peak parameters extracted by curve fitting are not distorted.

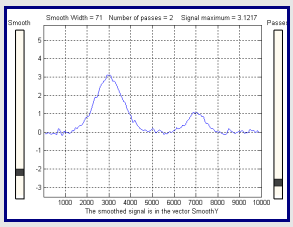
Video Demonstration. This 18-second, 3 MByte video (Smooth3.wmv) demonstrates the effect of triangular smoothing on a single Gaussian peak with a peak height of 1.0 and peak width of 200. The initial white noise amplitude is 0.3, giving an initial signal-to-noise ratio of about 3.3. An attempt to measure the peak amplitude and peak width of the noisy signal, shown at the bottom of the video, are initially seriously inaccurate because of the noise. As the smooth width is increased, however, the signal-to-noise ratio improves and the accuracy of the measurements of peak amplitude and peak width are improved. However, above a smooth width of about 40 (smooth ratio 0.2), the smoothing causes the peak to be shorter than 1.0 and wider than 200, even though the signal-to-noise

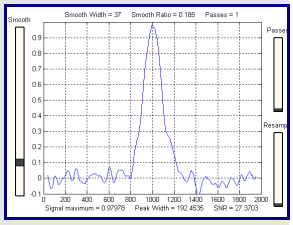
ratio continues to improve as the smooth width is increased. (This demonstration was created in Matlab 6.5 using "Interactive Smoothing for Matlab" module).

SPECTRUM includes rectangular and triangular smoothing functions for any number of points.

Smoothing in Matlab. The user-defined function fastsmooth implements all the types of smooths discussed above. (If you are viewing this document on-line, you can ctrl-click on this link to inspect the code). Fastsmooth is a Matlab function of the form **`s=fastsmooth(a,w,type,edge)`**. The argument "a" is the input signal vector; "w" is the smooth width; "type" determines the smooth type: type=1 gives a rectangular (sliding-average or boxcar); type=2 gives a triangular (equivalent to 2 passes of a sliding average); type=3 gives a pseudo-Gaussian (equivalent to 3 passes of a sliding average). The argument "edge" controls how the "edges" of the signal (the first w/2 points and the last w/2 points) are handled. If edge=0, the edges are zero. (In this mode the elapsed time is independent of the smooth width. This gives the fastest execution time). If edge=1, the edges are smoothed with progressively smaller smooths the closer to the end. (In this mode the execution time increases with increasing smooth widths). The smoothed signal is returned as the vector "s". (You can leave off the last two input arguments: **`fastsmooth(Y,w,type)`** smooths with edge=0 and **`fastsmooth(Y,w)`** smooths with type=1 and edge=0). Compared to convolution-based smooth algorithms, fastsmooth typically gives much faster execution times, especially for large smooth widths; it can smooth a 1,000,000 point signal with a 1,000 point sliding average in less than 0.1 second.

Interactive Smoothing for Matlab is a Matlab module for interactive smoothing for time-series signals, with sliders that allow you to adjust the smoothing parameters continuously while observing the effect on your signal dynamically. Can be used with any smoothing function. Includes a self-contained interactive demo of the effect of smoothing on peak height, width, and signal-to-noise ratio. If you have access to that software, you may download the complete set of Matlab Interactive Smoothing m-files, InteractiveSmoothing.zip, (12 Kbytes) so that you can experiment with all the variables at will and try out this technique on your own signal. Run SmoothSliderTest.m to see how it works.





Differentiation

The symbolic differentiation of functions is a topic that is introduced in all elementary Calculus courses. The numerical differentiation of digitized signals is an application of this concept that has many uses in analytical signal processing. The first derivative of a signal is the rate of change of y with x, that is, dy/dx, which is interpreted as the *slope* of the tangent to the signal at each point. Assuming that the x-interval between adjacent points is constant, the simplest algorithm for computing a first derivative is:

$$Y_j' = \frac{Y_{j+1} - Y_j}{X_{j+1} - X_j} = \frac{Y_{j+1} - Y_j}{\Delta X} \qquad X_j' = \frac{X_{j+1} + X_j}{2}$$ (for 1< j <n-1).

where X'_j and Y'_j are the X and Y values of the j^{th} point of the derivative, n = number of points in the signal, and ΔX is the difference between the X values of adjacent data points. A commonly used variation of this algorithm computes the average slope between three adjacent points:

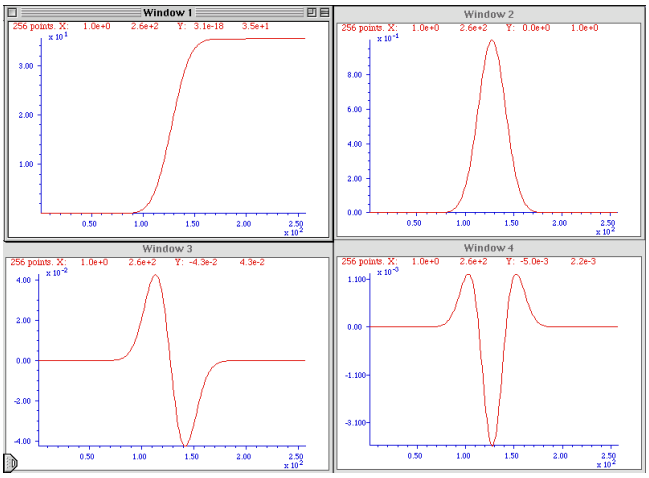
$$Y_j' = \frac{Y_{j+1} - Y_{j-1}}{2\Delta X} \qquad X_j' = X_j$$ (for 2 < j <n-1).

The *second derivative* is the derivative of the derivative: it is a measure of the *curvature* of the signal, that is, the rate of change of the slope of the signal. It can be calculated by applying the first derivative calculation twice in succession. The simplest algorithm for direct computation of the second derivative in one step is

$$Y_j'' = \frac{Y_{j+1} - 2Y_j + Y_{j-1}}{\Delta X^2} \qquad X_j' = X_j$$ (for 2 < j <n-1).

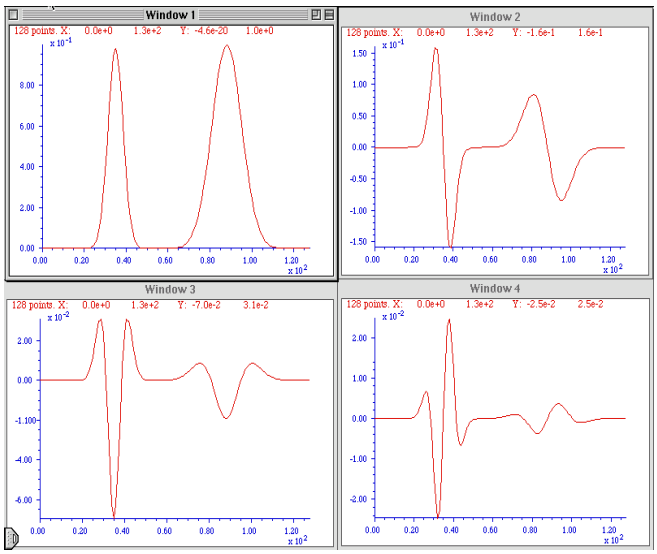
Similarly, higher derivative orders can be computed using the appropriate sequence of coefficients: for example +1, -2, +2, -1 for the third derivative and +1, -4, +6, -4, +1 for the 4^{th} derivative, although these derivatives can also be computed simply by taking successive lower order derivatives.

Basic Properties of Derivative Signals



The figure on the left shows the results of the successive differentiation of a computer-generated signal. The signal in each of the four windows is the first derivative of the one before it; that is, Window 2 is the first derivative of Window 1, Window 3 is the first derivative of Window 2, Window 3 is the *second* derivative of Window 1, and so on. You can predict the shape of each signal by recalling that the derivative is simply the slope of the original signal: where a signal slopes up, its derivative is positive; where a signal slopes down, its derivative is negative; and where a signal has zero slope, its derivative is zero. The sigmoidal signal shown in Window 1 has an *inflection point* (point where where the slope is maximum) at the center of the x axis range. This corresponds to the *maximum* in its first derivative (Window 2) and to the *zero-crossing* (point where the signal crosses the x-axis going either from positive to negative or *vice versa*) in the second derivative in Window 3. This behavior can be useful for precisely locating the inflection point in a sigmoid signal, by computing the location of the zero-crossing in its

second derivative. Similarly, the location of maximum in a peak-type signal can be computed precisely by computing the location of the zero-crossing in its first derivative.



Another important property of the differentiation of peak-type signals is the effect of the peak width on the amplitude of derivatives. The figure on the left shows the results of the successive differentiation of two computer-generated Gaussian bands. The two bands have the same amplitude (peak height) but one of them is exactly twice the width of the other. As you can see, the *wider* peak has the *smaller* derivative amplitude, and the effect becomes more noticeable at higher derivative orders. In general, it is found that that the amplitude of the n^{th} derivative of a peak is inversely proportional to the n^{th} power of its width. Thus differentiation in effect discriminates against wider peaks and the higher the order of differentiation the greater the discrimination. This behavior can be useful in quantitative analytical applications for detecting peaks that are superimposed on and obscured by stronger but broader background peaks.

Applications of Differentiation

A simple example of the application of differentiation of experimental signals is shown in Figure 5. This signal is typical of the type of signal recorded in amperometric titrations and some kinds of thermal analysis and kinetic experiments: a series of straight line segments of different slope. The objective is to determine how many segments there are, where the breaks between then fall, and the slopes of each segment. This is difficult to do from the raw data, because the slope differences are small and the resolution of the computer screen display is limiting. The task is much simpler if the first derivative (slope) of the signal is calculated (Figure 5, right). Each segment is now clearly seen as a separate step whose height (y-axis value) is the slope. The y-axis now takes on the units of dy/dx. Note that in this example the steps in the derivative signal are not completely flat, indicating that the line segments in the original signal were not perfectly straight. This is most likely due to random noise in the original signal. Although this noise was not particularly evident in the original signal, it is more noticeable in the derivative

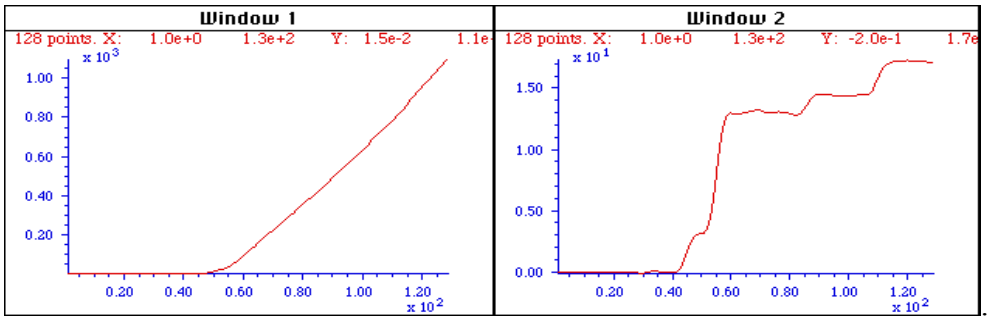


Figure 5. *The signal on the left seems to be a more-or-less straight line, but its numerically calculated* ***derivative*** *(dx/dy), plotted on the right, shows that the line actually has several approximately straight-line segments with distinctly different slopes and with well-defined breaks between each segment.*

It is commonly observed that differentiation degrades signal-to-noise ratio, unless the differentiation algorithm includes smoothing that is carefully optimized for each application. Numerical algorithms for differentiation are as numerous as for smoothing and must be carefully chosen to control signal-to-noise degradation.

A classic use of second differentiation in chemical analysis is in the location of endpoints in potentiometric titration. In most titrations, the titration curve has a sigmoidal shape and the endpoint is indicated by the *inflection point*, the point where the slope is maximum and the curvature is zero. The first derivative of the titration curve will therefore exhibit a *maximum* at the inflection point, and the second derivative will exhibit a *zero-crossing* at that point. Maxima and zero crossings are usually much easier to locate precisely than inflection points.

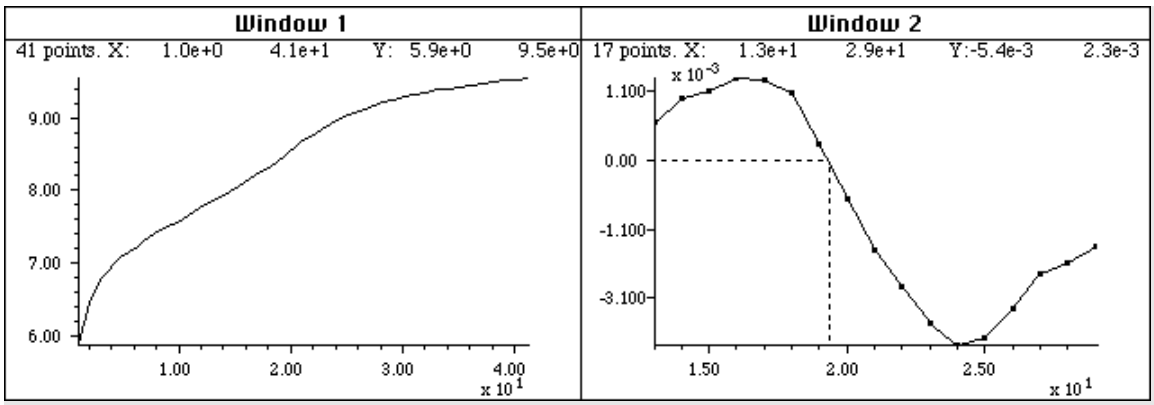


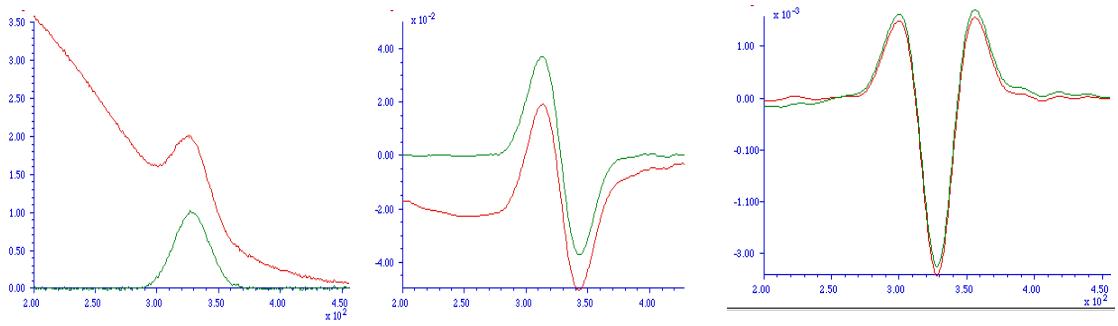
Figure 6 *The signal on the left is the pH titration curve of a very weak acid with a strong base, with volume in mL on the X-axis and pH on the Y-axis. The endpoint is the point of greatest slope; this is also an inflection point, where the curvature of the signal is zero. With a weak acid such as this, it is difficult to locate this point precisely from the original titration curve. The endpoint is much more easily located in the* ***second derivative****, shown on the right, as the the zero crossing.*

Figure 6 shows a pH titration curve of a very weak acid with a strong base, with volume in mL on the X-axis and pH on the Y-axis. The volumetric equivalence point (the "theoretical" endpoint) is 20 mL. The endpoint is the point of greatest slope; this is also an inflection point, where the curvature of the signal is zero. With a weak acid such as this, it is difficult to locate this point precisely from the original titration curve. The second derivative of the curve is shown in Window 2 on the right. The zero crossing of the second derivative corresponds to the endpoint and is much more precisely measurable. Note that in the second derivative plot, both the x-axis and the y-axis scales have been expanded to show the zero crossing point more clearly. The dotted lines show that the zero crossing falls at about 19.4 mL, close to the theoretical value of 20 mL.

Derivative Spectroscopy

In spectroscopy, the differentiation of spectra is a widely used technique, particularly in infra-red, u.v.-visible absorption, fluorescence, and reflectance spectrophotometry, referred to as *derivative spectroscopy*. Derivative methods have been used in analytical spectroscopy for three main purposes: (a) spectral discrimination, as a qualitative fingerprinting technique to accentuate small structural differences between nearly identical spectra; (b) spectral resolution enhancement, as a technique for increasing the apparent resolution of overlapping spectral bands in order to more easily determine the number of bands and their wavelengths; (c) quantitative analysis, as a technique for the correction for irrelevant background absorption and as a way to facilitate multicomponent analysis. (Because differentiation is a linear technique, the amplitude of a derivative is proportional to the amplitude of the original signal, which allows quantitative analysis applications employing any of the standard calibration techniques). Most commercial spectrophotometers now have built-in derivative capability. Some instruments are designed to measure the spectral derivatives optically, by means of dual wavelength or wavelength modulation designs.

Because of the fact that the amplitude of the n^{th} derivative of a peak-shaped signal is inversely proportional to the n^{th} power of the width of the peak, differentiation may be employed as a general way to discriminate against broad spectral features in favor of narrow components. This is the basis for the application of differentiation as a method of correction for background signals in quantitative spectrophotometric analysis. Very often in the practical applications of spectrophotometry to the analysis of complex samples, the spectral bands of the analyte (i.e. the compound to be measured) are superimposed on a broad, gradually curved background. Background of this type can be reduced by differentiation.



This is illustrated by the figure on the left, which shows a simulated UV spectrum (absorbance vs wavelength in nm), with the green curve representing the spectrum of the pure analyte and the red line representing the spectrum of a mixture containing the analyte plus other compounds that give rise to the large sloping background absorption. The first derivatives of these two signals are shown in the center; you can see that the difference between the pure analyte spectrum (green) and the mixture spectrum (red) is reduced. This effect is considerably enhanced in the second derivative, shown on the right. In this case the spectra of the pure analyte and of the mixture are almost identical. In order for this technique to work, it is necessary that the background absorption be broader (that is, have lower curvature) than the analyte spectral peak, but this turns out to

be a rather common situation. Because of their greater discrimination against broad background, second (and sometimes even higher-order) derivatives are often used for such purposes.

It is sometimes (mistakenly) said that differentiation "increases the sensitivity" of analysis. You can see how it would be tempting to say something like that by inspecting the three figures above; it does seems that the signal amplitude of the derivatives is greater (at least graphically) than that of the original analyte signal. However, it is not valid to compare the amplitudes of signals and their derivatives because they have different units. The units of the original spectrum are *absorbance*; the units of the first derivative are *absorbance per nm*, and the the units of the second derivative are *absorbance per nm*^{2}. You can't compare *absorbance* to *absorbance per nm* any more than you can compare *miles* to *miles per hour*. (It's meaningless, for instance, to say that *30 miles per hour* is greater than *20 miles*.) You can, however, compare the *signal-to-background ratio* and the *signal-to-noise ratio*. For example, in the above example, it would be valid to say that the signal-to-background ratio is increased in the derivatives.

Trace Analysis

One of the widest uses of the derivative signal processing technique in practical analytical work is in the measurement of small amounts of substances in the presence of large amounts of potentially interfering materials. In such applications it is common that the analytical signals are weak, noisy, and superimposed on large background signals. Measurement precision is often degraded by sample-to-sample baseline shifts due to non-specific broadband interfering absorption, non-reproducible cuvette positioning, dirt or fingerprints on the cuvette walls, imperfect cuvette transmission matching, and solution turbidity. Baseline shifts from these sources are usually either wavelength-independent (light blockage caused by bubbles or large suspended particles) or exhibit a weak wavelength dependence (small-particle turbidity). Therefore it can be expected that differentiation will in general help to discriminate relevant absorption from these sources of baseline shift. An obvious benefit of the suppression of broad background by differentiation is that *variations* in the background amplitude from sample to sample are also reduced. This can result in improved precision or measurement in many instances, especially when the analyte signal is small compared to the background and if there is a lot of uncontrolled variability in the background. An example of the improved ability to detect trace component in the presence of strong background interference is shown in Figure 7.

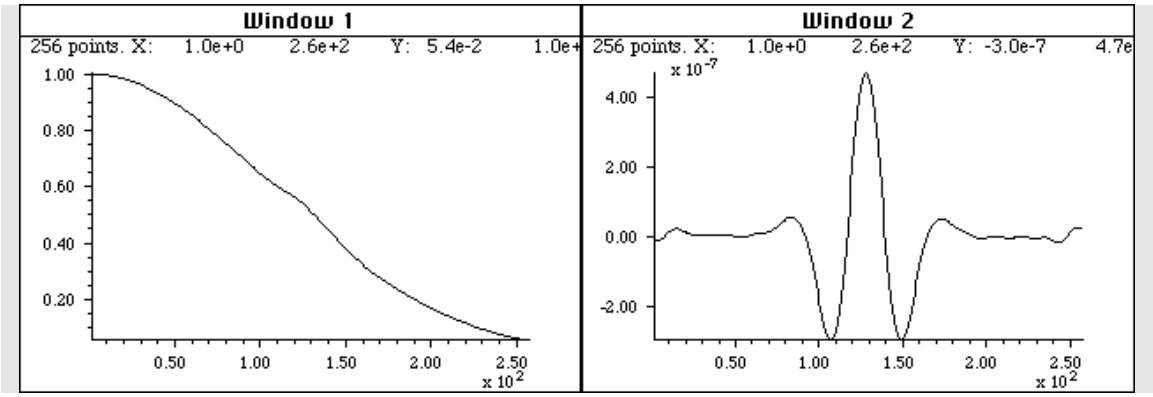


***Figure** 7. The spectrum on the left shows a weak shoulder near the center due to a small concentration of the substance that is to be measured (e.g. the active ingredient in a pharmaceutical preparation). It is difficult to measure the intensity of this peak because it is obscured by the strong background caused by other substances in the sample. The **fourth derivative** of this spectrum is shown on the right. The background has been almost completely suppressed and the analyte peak now stands out clearly, facilitating measurement.*

The spectrum on the left shows a weak shoulder near the center due to the analyte. The signal-to-noise ratio is very good in this spectrum, but in spite of that the broad, sloping background obscures the peak and makes quantitative measurement very difficult. The fourth derivative of this spectrum is shown on the right. The background has been almost completely suppressed and the analyte peak now stands out clearly, facilitating measurement. An even more dramatic case is shown in Figure 8. This is essentially the same spectrum as in Figure 7, except that the concentration of the analyte is lower. The question is: is there a detectable amount of analyte in this spectrum? This is quite impossible to say from the normal spectrum, but inspection of the fourth derivative (right) shows that the answer is *yes*. Some noise is clearly evident here, but nevertheless the signal-to-noise ratio is sufficiently good for a reasonable quantitative measurement.

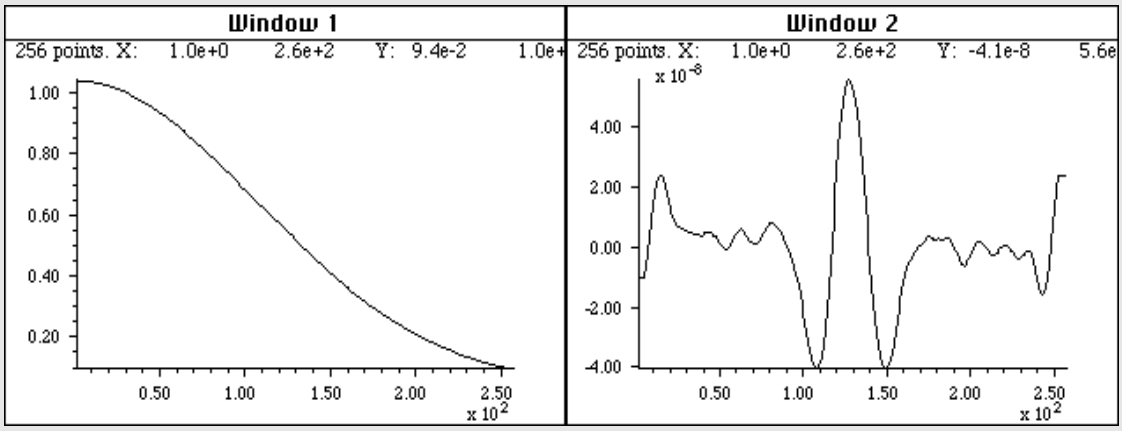
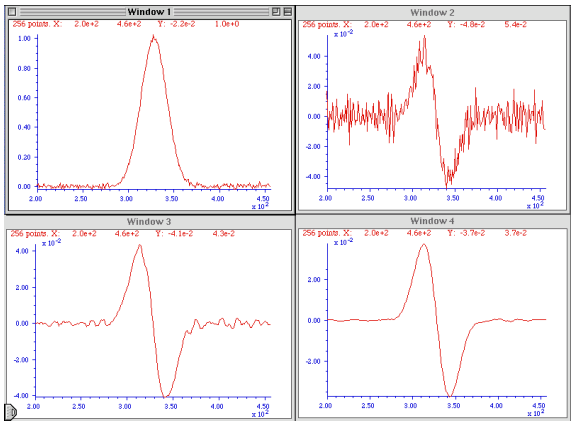


***Figure 8.** Similar to Figure 7, but in the case the peak is so weak that it can not even be seen in the spectrum on the left. The fourth derivative (right) shows that a peak is still there, but much reduced in amplitude (note the smaller y-axis scale).*

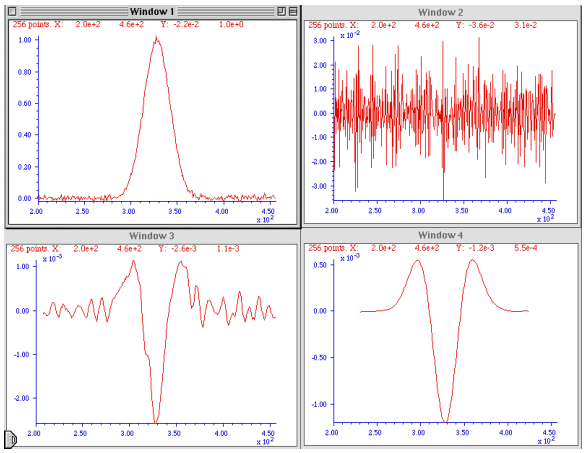
This use of signal differentiation has become widely used in quantitative spectroscopy, particularly for quality control in the pharmaceutical industry. In that application the analyte would typically be the active ingredient in a pharmaceutical preparation and the background interferences might arise from the presence of fillers, emulsifiers, flavoring or coloring agents, buffers, stabilizers, or other excipients. Of course, in trace analysis applications, care must be taken to optimize signal-to-noise ratio of the instrument as much as possible.

The Importance of Smoothing Derivatives

For the successful application of differentiation in quantitative analytical applications, it

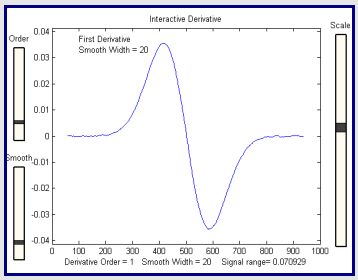


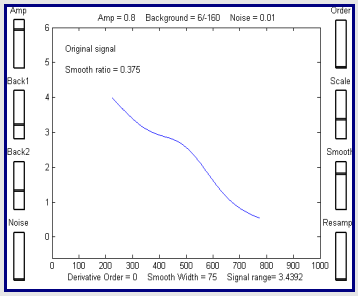
is essential to use differentiation in combination with sufficient smoothing, in order to optimize the signal-to-noise ratio. This is illustrated in the figure on the left. Window 1 shows a Gaussian band with a small amount of added noise. Windows 2, 3, and 4, show the first derivative of that signal with increasing smooth widths. As you can see, without sufficient smoothing, the signal-to-noise ratio of the derivative can be substantially poorer than the original signal. However, with adequate amounts of smoothing, the signal-to-noise ratio of the smoothed derivative can be better than that of the unsmoothed original. This effect is even more striking in the second derivative, as shown below. In this case, the signal-to-noise ratio of the unsmoothed second derivative (Window 2) is so poor you can not even see the signal visually. It makes no difference whether the smooth operation is applied before or after the differentiation. What is important, however, is the nature of the smooth, its smooth ratio (ratio of the smooth width to the width of the original peak), and the number of times the signal is smoothed. The optimum values of smooth ratio for derivative signals is approximately 0.5 to 1.0. For a first derivative, two applications of a simple rectangular smooth or one application of a triangular smooth is adequate. For a second derivative, three applications of a simple rectangular smooth or two applications of a triangular smooth is adequate. The general rule is: for the n^{th} derivative, use at least n+1 applications of rectangular smooth (or half that number of triangular smooths). Such heavy amounts of smoothing result in substantial attenuation of the derivative amplitude; in the figure on the right above, the amplitude of the most heavily smoothed derivative (in Window 4, above) is much less than its less-smoothed version (Window 3). However, this won't be a problem, as long as the standard (analytical) curve is prepared using the exact same derivative, smoothing, and measurement procedure as is applied to the unknown samples. (Because differentiation and smoothing are both linear techniques, the amplitude of a smoothed derivative is proportional to the amplitude of the original signal, which allows quantitative analysis applications employing any of the standard calibration techniques).



SPECTRUM includes first and second derivative functions, which can be applied successively to compute derivatives of any order.

Differentiation in Matlab





Differentiation functions such as described above can easily be created in Matlab. Some simple examples that you can download include: deriv, a first derivative using the 2-point central-difference method, deriv2, a simple second derivative using the 3-point central-difference method, a third derivative deriv3 using a 4-point formula, and deriv4, a 4th derivative using a 5-point formula. Each of these is a simple Matlab function of the form **`d=deriv(a)`**; the input argument is a signal vector "a", and the differentiated signal is returned as the vector "d". If you are viewing this document on-line, you can ctrl-click on these links to inspect the code.

Interactive Derivative for Matlab is a collection of Matlab functions and scripts for interactive differentiation of time-series signals, with sliders that allow you to adjust the derivative order, smooth width, and scale expansion continuously while observing the effect on your signal dynamically. Requires Matlab 6.5. Ctrl-click here to download the ZIP file "InteractiveDerivative.zip" that also includes supporting functions, self-contained demos to show how it works. Run InteractiveDerivativeTest to see how it works. Also includes DerivativeDemo.m, which demonstrates the application of differentiation to the detection of peaks superimposed on a strong, variable background. Generates a signal peak, adds random noise and a variable background, then differentiates and smooths it, and measures the signal range and signal-to-noise ratio (SNR). Interactive sliders allow you to control all the parameters. This was used to create the video demonstration DerivativeBackground2.wmv.

Derivative-based Peak Finding and Measurement in Matlab

This section describes a fast, customizable Matlab routine for locating and measuring the peaks in noisy time-series data sets. It detects peaks by looking for downward zero-crossings in the smoothed first derivative. Determines the position, height, and width of each peak by least-squares curve-fitting. (This is useful primarily for signals that have several data points in each peak, not for spikes that have only one or two points). It can find and measure 1000 peaks in a 1,000,000 point signal in 8 seconds.

The routine is available in three different versions: (1) the basic findpeaks function (findpeaks.m); (2) an interactive keypress-operated function (ipeak.m) for adjusting the peak detection criteria in real-time to optimize for any particular peak type; and (3) a script using mouse-controlled sliders for interactive control. These are all detailed below. Click here to download the ZIP file "Peakfinder.zip" that includes supporting functions, several self-contained demos to show how it works, and two interactive versions. You can also download it from the Matlab File Exchange.

Resolution enhancement

Figure 9 shows a spectrum that consists of several poorly-resolved (that is, partly overlapping) bands. The extensive overlap of the bands makes the accurate measurement of their intensities and positions impossible, even though the signal-to-noise ratio is very good. Things would be easier if the bands were more completely resolved, that is, if the bands were narrower.

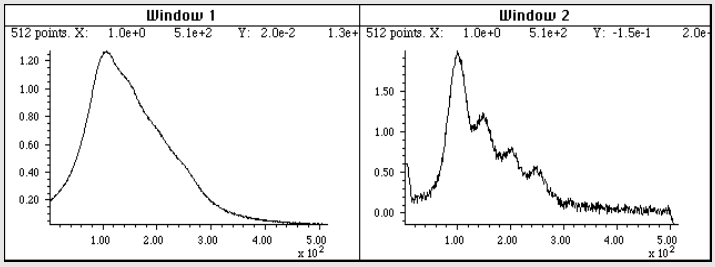


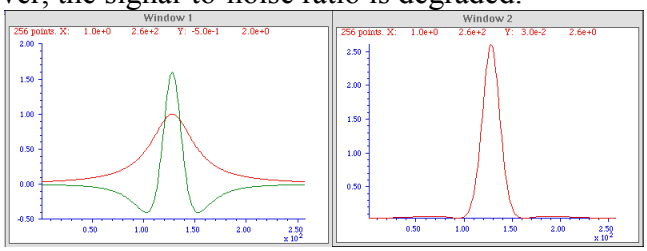
Figure 9. *A* ***resolution enhancement*** *algorithm has been applied to the signal on the left to artificially improve the apparent resolution of the peaks. In the resulting signal, right, the component bands are narrowed so that the intensities and positions can be measured.*

Here use can be made of *resolution enhancement* algorithms to artificially improve the apparent resolution of the peaks. One of the simplest such algorithms is based on the weighted sum of the original signal and the negative of its second derivative.

$$R_j = Y_j - kY_j''$$

where R_j is the resolution-enhanced signal, Y is the original signal, Y" is the second derivative of Y, and k is a user-selected weighting factor. It is left to the user to select the weighting factor k which gives the best trade-off between resolution enhancement, signal-to-noise degradation, and baseline undershoot. The optimum choice depends upon the width, shape, and digitization interval of the signal. (As a starting point, a reasonable value for k is $w^2/25$ for peaks of Gaussian shape, or $w^2/6$ for peaks of Lorentzian shape, where w is the number of data points in the half-width of the component peaks).

The result of the application of this algorithm is shown on the right in Figure 9. The component bands have been artificially narrowed so that the intensities and positions can be measured. However, the signal-to-noise ratio is degraded.



Here's how it works. The figure shows, in Window 1, a computer-generated peak (with a Lorentzian shape) in red, superimposed on the *negative* of its second derivative in green).

The second derivative is amplified (by multiplying it by an adjustable constant) so that the negative sides of the inverted second derivative (from approximately X = 0 to 100 and from X = 150 to 250) are a mirror image of the sides of the original peak over those regions. In this way, when the original peak is added to the inverted second derivative, the two signals will approximately cancel out in the two side regions but will reinforce each other in the central region (from X = 100 to 150). The result, shown in Window 2, is a substantial (about 50%) reduction in the width, and an increase in height, of the peak. This works best with Lorentzian-shaped peaks; with Gaussian-shaped peaks, the resolution enhancement is less dramatic (only about 20%). An interesting property of this procedure is that it does not change the total peak area (that is, the area under the peak) because the total area under the curve of the derivative of a peak-shaped signal is zero (the area under the negatives lobes cancels the area under the positive lobes).

Note: Another technique that can increase the resolution of overlapping peaks is deconvolution, which is applicable when the broadening function responsible for the overlap of the peaks is known. Deconvolution of the broadening function from the broadened peaks is in principle capable of extracting the underlying peaks shapes, whereas this resolution enhancement technique can not be expected to do that.

SPECTRUM includes this simple resolution-enhancement algorithm, with adjustable weighting factor and derivative smoothing width.

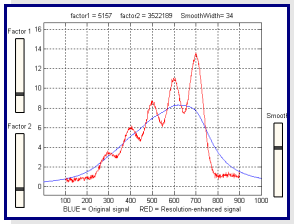
Resolution enhancement in Matlab. The user-defined Matlab function enhance.m (ctrl-click on this link to inspect the code) uses a slightly more advanced algorithm that extends the above approach by adding in a small amount of the 4th derivative of the signal:

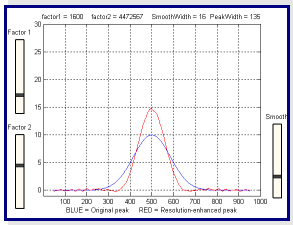
$$R = Y - k_2 Y'' + k_4 Y''''$$

This function has the form:

```
Enhancedsignal=enhance(signal,k2,k4,SmoothWidth),
```

where `signal` is the original signal vector, the arguments `k2` and `k4` are 2nd and 4th derivative weighting factors, and `SmoothWidth` is the width of the built-in smooth. The resolution-enhanced signal is returned in the vector `Enhancedsignal`.





The Interactive Resolution Enhancement for Matlab has sliders that allow you to adjust the resolution enhancement parameters (2nd derivative factor k_2, 4th derivative factor k_4, and the smooth width) while observing the effect on the signal output dynamically. It can handle signals of virtually any length, limited only by the memory in your computer. Requires Matlab 6.5. (If you have access to Matlab, you may download a set of Matlab resolution-enhancement m-files (16 Kbytes), ResolutionEnhancement.zip, so that you can experiment with all the variables at will and try out this technique on your own signals).

Video Demonstration. This 15-second, 1.7 MByte video (`ResEnhance3.wmv`) demonstrates the Matlab interactive resolution enhancement function InteractiveResEnhance.m. The signal consists of four overlapping, poorly-resolved Lorentzian bands. First, the 2nd derivative factor (Factor 1) is adjusted, then the 4th derivative factor (Factor 2) is adjusted, then the smooth

width (Smooth) is adjusted, and finally the Factor 2 is tweaked again.

Harmonic analysis and the Fourier Transform

Some signals exhibit periodic components that repeat at fixed intervals throughout the signal, like a sine wave. It is often useful to describe the amplitude and frequency of such periodic components exactly. Actually, it is possible to analyze any arbitrary set of data into periodic components, whether or not the data appear periodic. Harmonic analysis is conventionally based on the *Fourier transform*, which is a way of expressing a signal as a sum of sine and cosine waves. It can be shown that any arbitrary discretely sampled signal can be described completely by the sum of a finite number of sine and cosine components whose frequencies are 0,1,2,3 ... n/2 times the fundamental frequency $f=1/n\Delta x$, where Δx is the interval between adjacent x-axis values and n is the total number of points. The Fourier transform is simply the coefficients of these sine and cosine components.

The concept of the Fourier transform is involved in two very important instrumental methods in chemistry. In Fourier transform infrared spectroscopy (FTIR), the Fourier transform of the spectrum is measured directly by the instrument, as the interferogram formed by plotting the detector signal vs mirror displacement in a scanning Michaelson interferometer. In Fourier transform nuclear magnetic resonance spectroscopy (FTNMR), excitation of the sample by an intense, short pulse of radio frequency energy produces a free induction decay signal that is the Fourier transform of the resonance spectrum. In both cases the spectrum is recovered by inverse Fourier transformation of the measured signal.

The *power spectrum* is a simple way of showing the total amplitude at each of these frequencies; it is calculated as the square root of the sum of the squares of the coefficients of the sine and cosine components.

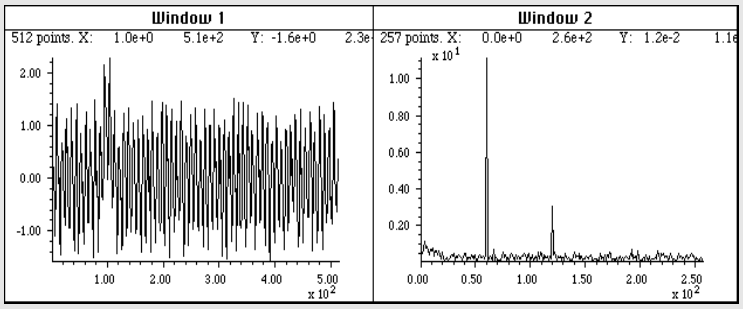


***Figure 10**. The signal on the left (x = time; y = voltage), which was expected to contain a single peak, is clearly very noisy. The **power spectrum** of this signal (x-axis = frequency in Hz) shows a strong component at 60 Hz, suggesting that much of the noise is caused by stray pick-up from the 60 Hz power line. The smaller peak at 120 Hz (the second harmonic of 60 Hz) probably comes from the same source.*

A signal with *n* points gives a power spectrum with only (n/2)+1 points. The x-axis is the *harmonic number.* The first point (x=0) is the zero-frequency (constant) component. The second point (x=1) corresponds to the fundamental frequency, the next point to twice the fundamental frequency, etc. An example of a practical application of the use of the power

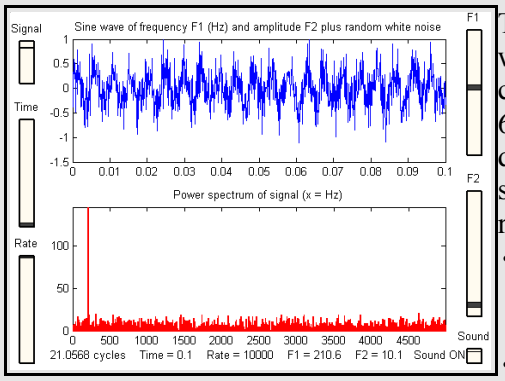
spectrum as a diagnostic tool is shown in Figure 10.

In the example shown here, the signal is a time-series signal with time as the independent variable. More generally, it is also possible to compute the Fourier transform and power spectrum of any signal, such as an optical spectrum, where the independent variable might be wavelength or wavenumber, or an electrochemical signal, where the independent variable might be volts. In such cases the units of the x-axis of the power spectrum are simply the reciprocal of the units of the x-axis of the original signal (e.g. nm^{-1} for a signal whose x-axis is in nm).

SPECTRUM includes a power spectrum function, as well as forward and reverse Fourier transformation.

Matlab has built-in functions for computing the Fourier transform (FFT and IFFT) and the power spectrum (powerspectrum(f,T)).

Interactive Power Spectrum Demo for Matlab



This is digital signal generator (simulator) for Matlab, with power spectrum display, sliders for real time control, and audio waveform output. Requires Matlab 6.5. This program is useful for teaching and demonstrating the power spectra of different types of signals and the effect of signal duration and sampling rate.

- The **Signal** slider selects from 10 different pre-programmed signals*. (You can change the signal definitions in RedrawSpectrum.m).
- The **Time** slider controls the total duration of signal (seconds).
- The **Rate** slider controls the sample rate (Hz).
- The **F1** and **F2** sliders control the global variables f1 and f2 which are used to control the frequency, phase, time, and/or amplitude of the different pre-programmed signals as described below.
- The **Sound** slider has only two settings: OFF (down) and ON (up). When Sound is ON, each time a slider is moved, the signal (in the variable "y") is sent to the Windows WAVE audio device. When Sound is OFF, no sound is played when a slider is moved. However, even when Sound is OFF, clicking on the Sound slider plays the signal once for each click.

If you are viewing this document on-line, you can ctrl-click here to download the ZIP file "PowerSpectrumDemo11.zip", which also includes supporting functions and self-contained demos to show how it works. You can also download it from the Matlab File Exchange.

* The 10 pre-programmed signals are: Sine wave of frequency f1 (Hz) and phase f2; Sine wave burst of frequency f1 (Hz) and duration f2; Square wave of frequency f1 (Hz) and phase f2; Sawtooth wave of frequency f1(Hz); 440 Hz carrier amplitude modulated by sine wave of frequency f1 (Hz) and amplitude f2; 440 Hz carrier frequency modulated by sine wave of frequency f1 (Hz) and amplitude f2; Sine wave of frequency f1 (Hz) modulated by Gaussian pulse of width f2; Random white noise; Sine wave of frequency f1 (Hz) and amplitude f2 plus random white noise; Sine wave sweep from 0 to f1 (Hz).

Convolution

Convolution is an operation performed on two signals which involves multiplying one signal by a delayed version of another signal, integrating or averaging the product, and repeating the process for different delays. Convolution is a useful process because it accurately describes some effects that occur widely in scientific measurements, such as the influence of a low-pass filter on an electrical signal or of a spectrometer on the shape of a spectrum.

In practice the calculation is usually performed by multiplication of the two signals in the Fourier domain. First, the Fourier transform of each signal is obtained. Then the two Fourier transforms are multiplied by the rules for complex multiplication and the result is then inverse Fourier transformed. Although this seems to be a round-about method, it turns out to be faster then the shift-and-multiply algorithm when the number of points in the signal is large. Convolution can be used as a very powerful and general algorithm for smoothing and differentiation. The example of Figure 11 shows how it can be used to predict the broadening effect of a spectrometer on an atomic line spectrum.

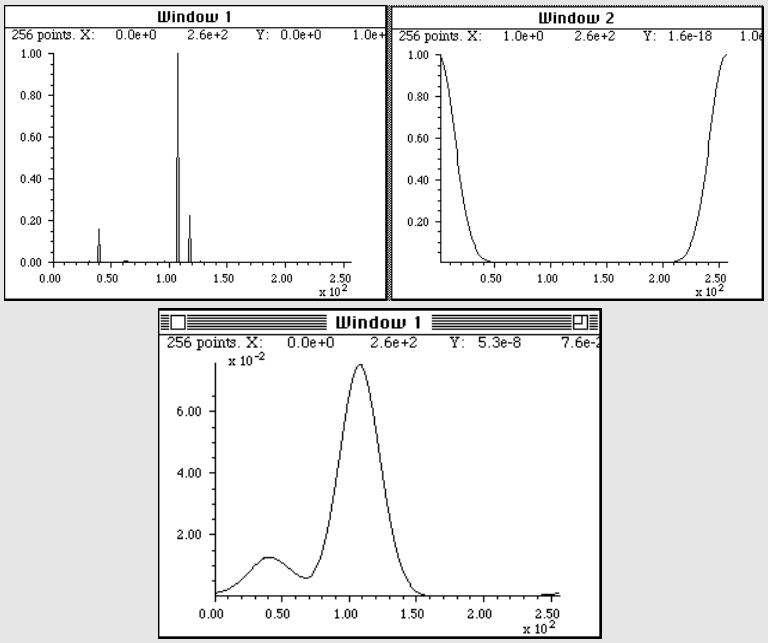


Figure 11. ***Convolution*** *is used here to determine how the atomic line spectrum in Window 1 (top left) will appear when scanned with a spectrometer whose slit function (spectral resolution) is described by the Gaussian function in Window 2 (top right). The Gaussian function has already been rotated so that its maximum falls at x=0. The resulting convoluted spectrum (bottom center) shows that the two lines near x=110 and 120 will not be resolved but the line at x=40 will be partly resolved.*

SPECTRUM includes convolution and auto-correlation (self-convolution) functions.
Matlab has a built-in function for convolution: conv. This function can be used to create very general type of filters and smoothing functions, such as sliding-average and triangular smooths.

Deconvolution

Deconvolution is the converse of convolution in the sense that division is the converse of multiplication. In fact, the deconvolution of one signal from another is usually performed by *dividing* the two signals in the Fourier domain. The practical significance of deconvolution is that it can be used as an artificial (i.e. computational) way to reverse the result of a convolution occurring in the physical domain, for example, to reverse the signal distortion effect of an electrical filter or of the finite resolution of a spectrometer.Two examples of the application of deconvolution are shown in Figures 12 and 13.

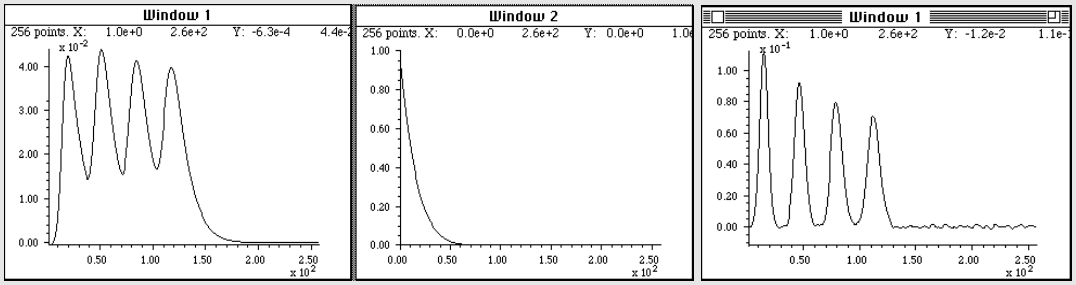


Figure 12. ***Deconvolution*** *is used here to remove the distorting influence of an exponential tailing response function from a recorded signal (Window 1, left) that is the result of an unavoidable RC low-pass filter action in the electronics. The response function (Window 2, center) is usually either calculated on the basis of some theoretical model or is measured experimentally as the output signal produced by applying an impulse (delta) function to the input of the system. The response function, with its maximum at x=0, is deconvoluted from the original signal . The result (right) shows a closer approximation to the real shape of the peaks; however, the signal-to-noise ratio is unavoidably degraded.*

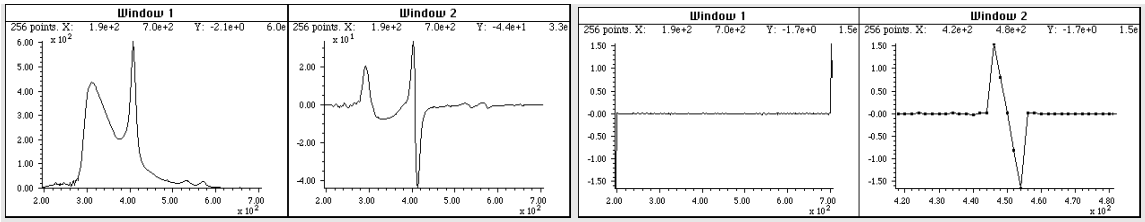


Figure 13. A different application of the deconvolution function is to reveal the nature of an unknown data transformation function that has been applied to a data set by the measurement instrument itself. In this example, Window 1 (top left) is a uv-visible absorption spectrum recorded from a commercial photodiode array spectrometer (X-axis: nanometers; Y-axis: milliabsorbance). Window 2 (top right) is the first derivative of this spectrum produced by an (unknown) algorithm in the software supplied with the spectrometer. The signal in the bottom left is the result of deconvoluting the derivative spectrum (top right) from the original spectrum (top left). This therefore must be the convolution function used by the differentiation algorithm in the spectrometer's software. Rotating and expanding it on the x-axis makes the function easier to see (bottom right). Expressed in terms of the smallest whole numbers, the convolution series is seen to be +2, +1, 0, -1, -2. This simple example of "reverse engineering" would make it easier to compare results from other instruments or to duplicate these result on other equipment.

When applying deconvolution to experimental data, to remove the effect of a known broadening or low-pass filter operator caused by the experimental system, a very serious signal-to-noise degradation commonly occurs. Any noise added to the signal by the system after the broadening or low-pass filter operator will be greatly amplified when the Fourier transform of the signal is divided by the Fourier transform of the broadening operator, because the high frequency components of the broadening operator (the denominator in the division of the Fourier transforms) are typically very small, resulting in a great amplification of high frequency noise in the resulting deconvoluted signal. This can be controlled but not completely eliminated by smoothing and by constraining the deconvolution to a frequency region where the signal has a sufficiently high signal-to-noise ratio.

Note: The term "deconvolution" is sometimes also used for the process of resolving or decomposing a set of overlapping peaks into their separate components by the technique of iterative least-squares curve fitting of a putative peak model to the data set. The process is actually conceptually distinct from deconvolution, because in deconvolution the underlying peak shape is unknown but the broadening function is assumed to be known; whereas in iterative least-squares curve fitting the underlying peak shape is assumed to be known but the broadening function is unknown.

SPECTRUM includes a deconvolution function. Matlab has built-in function for deconvolution: deconv.

Fourier filter

The Fourier filter is a type of filtering or smoothing function that is based on the frequency components of a signal. It works by taking the Fourier transform of the signal, then cutting off all frequencies above a user-specified limit, then inverse transforming the result. The assumption is made here that the frequency components of the signal fall predominantly at low frequencies and those of the noise fall predominantly at high frequencies. The user tries to find a cut-off frequency that will allow most of the noise to be eliminated while not distorting the signal significantly. An example of the application of the Fourier filter is given in Figure 14.

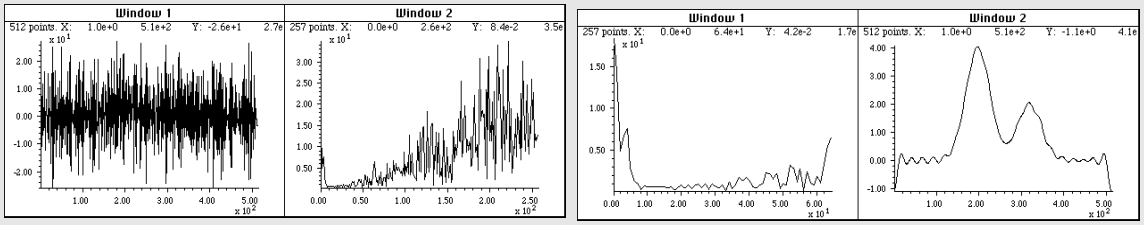


***Figure 14**. The signal at the far left seems to be only random noise, but its **power spectrum** (Window 2, second from left) shows that high-frequency components dominate the signal. The power spectrum is expanded in the X and Y directions to show more clearly the low-frequency region (Window 1, right). Working on the hypothesis that the components above the 20th harmonic are noise, the **Fourier filter** function can be used to delete the higher harmonics and to reconstruct the signal from the first 20 harmonics.*

The result (far right) shows the signal contains two bands at about x=200 and x=300 that are totally obscured by noise in the original signal.

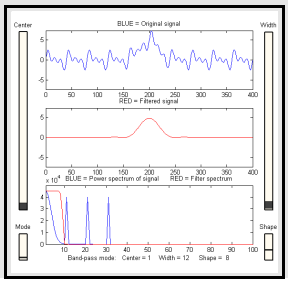
SPECTRUM includes a simple Fourier low-pass filter function with adjustable harmonic cut-off.

The user-defined Matlab function FouFilter.m is a more flexible Fourier filter that can serve as a lowpass, highpass, bandpass, or bandreject (notch) filter with variable cut-off rate. Has the form

```
ry=FouFilter(y,samplingtime,centerfrequency,frequencywidth,shape,mode)
```

where y is the time-series signal vector, 'samplingtime' is the total duration of sampled signal in sec, millisec, or microsec; 'centerfrequency' and 'frequencywidth' are the center frequency and width of the filter in Hz, KHz, or MHz, respectively; 'Shape' determines the sharpness of the cut-off. If shape = 1, the filter is Gaussian; as shape increases the filter shape becomes more and more rectangular. Set mode = 0 for band-pass filter, mode = 1 for band-reject (notch) filter. FouFilter returns the filtered signal in ry.

The Interactive Fourier Filter for Matlab function has sliders that allow you to adjust the Fourier filter parameters (center frequency, filter width, and cut-off rate) while observing the effect on the signal output dynamically. It can handle signals of virtually any length, limited only by the memory in your computer. Requires Matlab 6.5.



Integration and peak area measurement

The symbolic integration of functions and the calculation of definite integrals are topics that are introduced in elementary Calculus courses. The numerical integration of digitized signals finds application in analytical signal processing mainly as a method for measuring the areas under the curves of peak-type signals. Peak area measurements are very important in chromatography. Quantitation in chromatography is customarily carried out on the basis of peak area rather than peak height measurement. The reason for this is that peak area is less sensitive to the influence of peak broadening (dispersion) mechanisms. These broadening effects, which arise from many sources, cause chromatographic peaks to become shorter, broader, and more unsymmetrical, but have little effect on the total area under the peak. The peak area remains proportional to the total quantity of substance passing into the detector. Therefore peak area measurements are often found to be more reliable than peak height measurement.

The simple numeric integration of a digital, e.g. by Simpson's rule, will convert a series of peaks into a series of steps, the height of each of which is proportional to the area under that peak. This is a commonly used method in proton NMR spectroscopy, where the area under each peak or multiplet is proportional to the number of equivalent hydrogens responsible for that peak. But this works well only if the peaks are well separated from each other (e.g. well resolved).

In chromatographic analysis one often has the problem of measuring the the area under

the curve of the peaks when they are not well resolved or are superimposed on a background. For example, Figure 15 shows a series of four computer-synthesized Gaussian peaks that all have the same height, width, and area, but the separation between the peaks on the right is insufficient to achieve complete resolution.

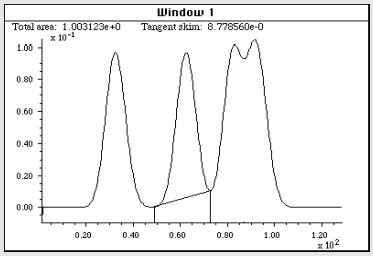
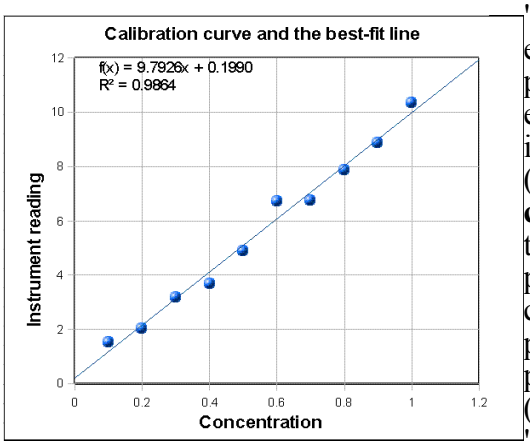


Figure 15. *Peak area measurement for overlapping peaks, using the perpendicular drop method.*

The classical way to handle this problem is to draw two vertical lines from the left and right bounds of the peak down to the x-axis and then to measure the total area bounded by the signal curve, the x-axis (y=0 line), and the two vertical lines. This is often called the the perpendicular drop method, and it is an easy task for a computer, although very tedious to do by hand. The idea is illustrated for the second peak from the left in Figure 15. The left and right bounds of the peak are usually taken as the valleys (minima) between the peaks. Using this method it is possible to estimate the area of the second peak in this example to an accuracy of about 0.3% and the second and third peaks to an accuracy of better than 4%, despite the poor resolution.

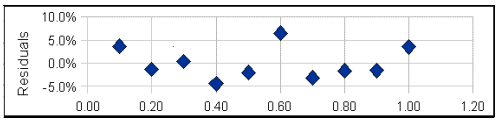
Curve fitting A: Linear Least Squares

The objective of curve fitting is to find a mathematical equation that describes a set of data and that is minimally influenced by random noise. The most common approach is the "linear least squares" method, also called "polynomial least squares", a well-known mathematical procedure for finding the coefficients of polynomial equations that are a "best fit" to a set of X,Y data. A polynomial equation expresses the dependent variable Y as a polynomial in the independent variable X, for example as a straight line (Y = **a** + **b**X, where **a** is the *intercept* and **b** is the *slope*), or a quadratic (Y = **a** + **b**X + **c**X^2), or a cubic (Y = **a** + **b**X + **c**X^2 + **d**X^3), or higher-order polynomial. In all these cases, Y is a linear function of the parameters **a**,**b**,**c**, and **d**. *This* is why this is called "linear" least-squares fit, *not* because the plot of X vs Y is linear. Only for the first-order polynomial is the plot of X vs Y linear. (Sometimes this type of curve fitting is called "curvilinear").



The graph shows a simple straight-line example (Y = **a** + **b**X), a small set of data where X = concentration and Y = instrument reading. The blue dots are the data points. The least-squares algorithm computes the values of **a** (intercept) and **b** (slope) of the straight

line that is a least-squares best fit to the data points. For this set of data, the slope is 9.7926 and the intercept is 0.199. The least squares procedure also calculates R^2, the *coefficient of determination*, also called the *correlation coefficient*, which is an indicator of the "goodness of fit". R^2 is exactly 1.0000 when the fit is perfect, less than that when the fit is imperfect. The closer to 1.0000 the better. The points do not all fall on the straignt line because of random noise in the data. (In this example, the "true" value of the slope is 10 and of the intercept is zero, so the presense of the noise caused this particular measurement of slope to be off by about 2%. Had there been a larger number of points in this data set, the calculated values of slope and intercept would probably have been better).

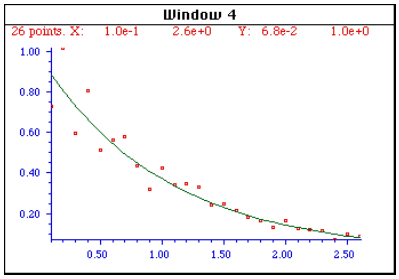


A simple way to evaluate a curve fit is to look at a plot of the "residuals", the differences between the y values in the original data and the y values computed by the fit equation. If the residuals are completely random and show no systematic variation, then the fit is as good as possible for that level of noise; but if the residual plot shows a regular curve, for example a U-shaped or an S-shaped curve, it may mean that another fitting function would be more appropriate.

Least-squares best fits can be calculated by some hand-held calculators, spreadsheets, and dedicated computer programs (see below for details). Although it is possible to estimate the best-fit straight line by visual estimation and a straightedge, the least-square method is more objective and easier to automate. (If you were to give this set of data to five different people and ask them to estimate the best-fit line visually, you'd get five slightly different answers, but it you gave the data set to five different computer programs, you'd get the exact same answer every time).

Transforming non-linear relationships

In some cases a fundamentally non-linear relationship can be transformed into a form that is amenable to polynomial curve fitting by means of a coordinate transformation (e.g. taking the log or the reciprocal of the data), and then the least-squares method can be applied to the resulting linear equation. For example, the points in Figure 16 are from the simulation of an exponential decay (X = time, Y = signal intensity) that has the mathematical form Y = **a** exp(**b**X), where **a** is the Y-value at X = 0 and **b** is the decay constant. In this particular example, **a** = 1 and **b** = -1. This is a fundamentally non-linear problem because Y is a non-linear function of the parameter **b**. However, by taking the natural log of both sides of the equation, we obtain ln(Y) = ln(**a**) + **b**X. In this equation, ln(Y) is a linear function of ln(**a**) and **b**, so it can be fit by the least squares method, from which values of **a** and **b** are easily calculated.



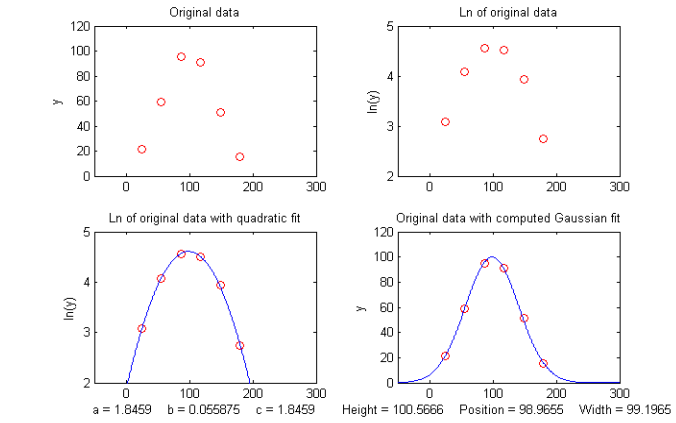
An exponential least-squares fit (solid line) applied to a noisy data set (points) in order to estimate the decay constant.

The best fit equation, shown by the green solid line in the figure, is Y = 0.9897 exp(- 0.98896 X), that is **a** = 0.9897 and **b** = -0.98896, very

close to the expected values. Thus, even in the presence of substantial random noise, it is possible to get good estimates of the parameters of the underlying equation. The most important requirement is that the model be good, that is, that the equation selected for the model accurately describes the underlying behavior of the system (except for noise). Often that is the most difficult aspect, because the underlying models are not always known with certainty.

Other examples of non-linear relationships that can be linearized by coordinate transformation include the logarithmic ($Y = \mathbf{a}\ \ln(\mathbf{b}X)$) and power ($Y=\mathbf{a}X^{\mathbf{b}}$) relationships. Methods of this type used to be very common back in the days before computers, when fitting anything but a straight line was difficult. It is still used today to extend the range of functional relationships that can be handled by common linear least-squares routines. (Only a few non-linear relationships can be handled this way, however. To fit any arbitrary custom function, you may have to resort to the more difficult Non-linear Iterative Curve Fitting method).

Another example of the use of transformation to convert a non-linear relationship into a form that is amenable to polynomial curve fitting is the use of the natural log (ln) transformation to convert a Gaussian peak, which has the fundamental functional form $\exp(-x^2)$, into a parabola of the form $-x^2$, which can be fit with a second order polynomial (quadratic) function ($y = \mathbf{a} + \mathbf{b}x + \mathbf{c}x^2$). All three parameters of the Gaussian (height, maximum position, and width) can be calculated from the three quadratic coefficients **a**, **b**, and **c**; the peak height is given by `exp(a-c*(b/(2*c))^2)`, the peak position by `-b/(2*c)`, and the peak half-width by `2.35703/(sqrt(2)*sqrt(-c))`.



An example is shown in the figure on the left. The signal is a Gaussian peak with a peak height of exactly 100 units, a peak position of 100 units, and a half-width of 100 units, but it is sampled only every 31 units on the x-axis. The resulting data set, shown by the red points in the upper left, has only 6 data points on the peak, too few for accurate measurement of peak height, position, and width. The maximum of those 6 points is at x=87 and has an amplitude of 95, not that close to the true values of peak position and height (100). However, taking the natural log of the data (upper right) produces a parabola that can be fit with a quadratic least-squares fit (shown by the blue line in the lower left). From the three coefficients of the quadratic fit we can calculate much more accurate values of the Gaussian peak parameters, shown at the bottom of the figure. The plot in the lower right shows the resulting Gaussian fit (in blue) displayed with the original data (red points). The accuracy is limited only by the noise in the data (about 1% in this example). This figure was created in Matlab, using this script. Note: in order for this method to work properly, the data set must not contain any zeros or negatives points, and the original Gaussian peak signal must have a zero baseline, that is, must tend to zero far from the peak center. In practice this means that any non-zero baseline must be subtracted from the data set before applying this method. (A more general approach to fitting Gaussian peaks, which works for data sets with zeros and negative numbers and also for data with multiple overlapping peaks, is the non-linear iterative curve fitting method).

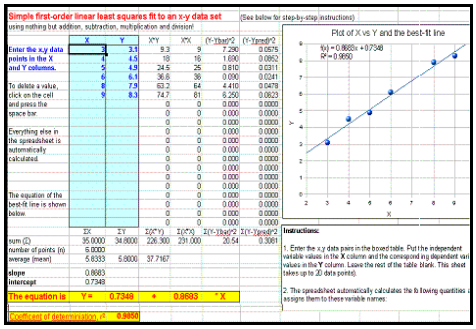
Note: In the curve fitting techniques described here and in the next two sections, there is no requirement that the x-axis interval between data points be uniform, as is the assumption in many of the other signal processing techniques previously covered. Curve fitting algorithms typically accept a set of arbitrarily-spaced x-axis values and a corresponding set of y-axis values.

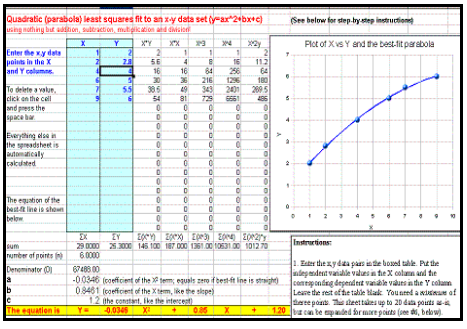
Math details

The least-squares best fit for an x,y data set can be computed using only basic arithmetic. Here are the relevant equations for computing the slope and intercept of the first-order best-fit equation, y = intercept + slope*x, as well as the coefficient of determination, "R2", which is an indicator of the "goodness of fit". (R2 is 1.0000 if the fit is perfect and less than that if the fit is imperfect):

```
n = number of x,y data points
sumx = Σx
sumy = Σy
sumxy = Σx*y
sumx2 = Σx*x
meanx = sumx / n
meany = sumy / n
slope = (n*sumxy - sumx*sumy) / (n*sumx2 - sumx*sumx
intercept = meany-(slope*meanx)
ssy = Σ(y-meany)^2
ssr = Σ(y-intercept-slope*x)^2
R2 = 1-(ssr/ssy)
```

Popular spreadsheets have facilities for computing polynomial least-squares curve fits. For example, the LINEST function in both Excel and OpenOffice Calc can be used to compute polynomial and other curvilinear least-squares fits. For some examples of applications to analytical chemistry, see Calibration Curve Fitting Methods in Absorption Spectroscopy and Error propagation in Analytical Calibration.

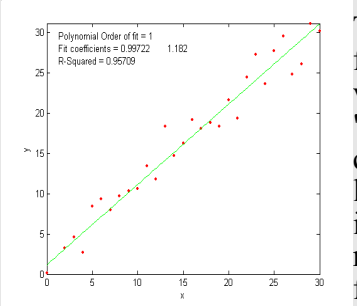




You can also download the spreadsheets pictured above, in OpenOffice Calc format, that automate the computation of those equations and also plot the data and the best-fit line, requiring only that you type in (or paste in) the x-y data. There is one spreadsheet for linear fits (http://terpconnect.umd.edu/~toh/spectrum/LeastSquares.odt) and also a version for quadratic (parabolic) fits (http://terpconnect.umd.edu/~toh/spectrum/QuadraticLeastSquares.ods). For the application to analytical calibration, there are specific versions of these spreadsheets that also calculate the concentrations of the unknowns (http://terpconnect.umd.edu/~toh/models/CalibrationCurve.html).

SPECTRUM includes least-squares curve fitting for polynomials of order 1 through 5, plus exponential, logarithmic, and power relationships.

Matlab has simple built-in functions for least-squares curve fitting: polyfit and polyval. For example, if you have a set of x,y data points in the vectors "x" and "y", then the coefficients for the least-squares fit are given by `coef=polyfit(x,y,n)`, where "n" is the order of the polynomial fit: n = 1 for a straight-line fit, 2 for a quadratic (parabola) fit, etc. For a straight-line fit (n=1), `coef(1)` is the slope ("b") and `coef(2)` is the intercept ("a"). For a quadratic fit (n=2), `coef(1)` is the x^2 term ("c"), `coef(2)` is the x term ("b") and `coef(3)` is the constant term ("a"). The fit equation can be evaluated using the function polyval, for example `fity=polyval(coef,x)`. This works for any order of polynomial fit ("n"). You can plot the data and the fitted equation together using the `plot` function: `plot(x,y,'ob',x,polyval(coef,x),'-r')`, which plots the data as blue circles and the fitted equation as a red line. You can plot the residuals by writing `plot(x,y-polyval(coef,x))`. (When the number of data points is small, you can get a smoother plot of the fitted equation, evaluated at more finely-divided values of x, by defining `xx=linspace(min(x),max(x));` and then using xx rather than x to evaluate and plot the fit: `plot(x,y,'ob',xx,polyval(coef,xx),'-r'))`.



The graph on the left was generated by the user-defined function plotfit(x,y,polyorder), which accepts the data in the vectors "x" and "y", fits it to a polynomial of order "polyorder", plots the data and the fit, and displays the fit coefficients and the goodness-of-fit measure R^2 in the upper left corner of the graph. R^2 is always between zero and one: it is 1.0 if the fit is perfect and zero if there is no systematic relation at all between x and y. (Ctrl-click to view this function).

Curve fitting B: Multicomponent Spectroscopy

The spectroscopic analysis of mixtures, when the spectra of the components overlap considerably, require special calibration methods based on a type of linear least squares called multiple linear regression. This method is widely used in multi-wavelength techniques such as diode-array, Fourier transform, and automated scanning spectrometers.

Definitions:

A = analytical signal — c1, c2 = species 1, species 2, etc.
ε = analytical sensitivity — λ1, λ2 = wavelength 1, wavelength 2, etc.
c = molar concentration — s1,s2 = sample 1, sample 2, etc.
s = number of samples — *n* = number of distinct chemical species in the mixture.
w = number of wavelengths at which signal is measured

Assumptions:

The analytical signal, A (such as absorbance in absorption spectroscopy, fluorescence intensity in fluorescence spectroscopy, and reflectance in reflectance spectroscopy.) is directly proportional to concentration, c. The proportionality constant, which is the slope of a plot of A vs c, is ε.

$$A = \varepsilon c$$

The total signal is the sum of the signals for each component in a mixture:

$$A_{total} = A_{c1} + A_{c2} + \ldots \text{ for all } n \text{ species.}$$

Classical Least Squares (CLS) calibration. This method is applicable to the quantitative analysis of a mixture of species when the spectra of the individual species are known. Measurement of the spectra of known concentrations of the separate species allows their analytical sensitivity ε at each wavelength to be determined. Then it follows that:

$$A_{\lambda 1}=\varepsilon_{c1,\lambda 1}\ c_{c1}+\varepsilon_{c2,\lambda 1}\ c_{c2}+\varepsilon_{c3,\lambda 1}\ c_{c3}+\ldots \text{ for all } n \text{ species.}$$

$$A_{\lambda 2}=\varepsilon_{c1,\lambda 2}\ c_{c1}+\varepsilon_{c2,\lambda 2}\ c_{c2}+\varepsilon_{c3,\lambda 2}\ c_{c3}+\ldots$$

and so on for all wavelengths. This set of linear equations is written compactly in matrix form:

$$\mathbf{A} = \boldsymbol{\varepsilon}\mathbf{C}$$

where **A** is the *w*-length vector of measured signals (i.e. the signal spectrum) of the mixture, $\boldsymbol{\varepsilon}$ is the *n* x *w* rectangular matrix of the known ε-values for each of the *n* species at each of the *w* wavelengths, and **C** is the *n*-length vector of concentrations of all the species.

If you have a sample solution containing unknown amounts of those species, then you measure its spectrum **A** and seek to calculate the concentration vector **C**. In order to solve the above matrix equation for **C**, the number of wavelengths *w* must be equal to or greater than the number of species *n*. If $w = n$, then we have a system of *n* equations in *n* unknowns which can be solved by pre-multiplying both sides of the equation by $\boldsymbol{\varepsilon}^{-1}$, the matrix inverse of $\boldsymbol{\varepsilon}$, and using the property that any matrix times its inverse is unity:

$$\mathbf{C} = \boldsymbol{\varepsilon}^{-1}\mathbf{A}$$

Because real experimental spectra are subject to random noise (e.g. photon noise and detector noise), the solution will be more precise if signals at a larger number of wavelengths are used, i.e. if $w > n$. But then the equation can not be solved by matrix inversion, because the $\boldsymbol{\varepsilon}$ matrix is a *w* x *n* matrix and a matrix inverse exists only for square matrices. A solution can be obtained in this case by pre-multiplying both sides of the equation by the expression $(\varepsilon^T\varepsilon)^{-1}\varepsilon^T$:

$$(\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon})^{-1}\boldsymbol{\varepsilon}^T\mathbf{A} = (\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon})^{-1}\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon}\mathbf{C} = (\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon})^{-1}(\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon})\mathbf{C}$$

But the quantity $(\varepsilon^T\varepsilon)^{-1}(\varepsilon^T\varepsilon)$ is a matrix times its inverse and is therefore unity. Thus:

$$\mathbf{C} = (\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon})^{-1}\boldsymbol{\varepsilon}^T\mathbf{A}$$

In this expression, $\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon}$ is a square matrix of order *n*, the number of species. In most practical applications, *n* is typically only 2 to 5, this is not a very big matrix to invert, no matter how many wavelengths are used. In general, the more wavelengths are used the more effectively the random noise will be averaged out (although it won't help to use wavelengths in spectral regions where none of the components produce analytical signals). The determination of the optimum wavelength region must usually be determined empirically.

Two extensions of the CLS method are commonly made. First, in order to account for baseline shift caused by drift, background, and light scattering, a column of 1s is added to the $\boldsymbol{\varepsilon}$ matrix. This has the effect of introducing into the solution an additional component with a flat spectrum; this is referred to as "background correction". Second, in order to account for the fact that the precision of measurement may vary with wavelength, it is common to perform a *weighted* least squares solution that de-emphasizes wavelength regions where precision is poor:

$$\mathbf{C} = (\boldsymbol{\varepsilon}^T\ \mathbf{V}^{-1}\boldsymbol{\varepsilon})^{-1}\ \boldsymbol{\varepsilon}^T\ \mathbf{V}^{-1}\ \mathbf{A}$$

where **V** is an *w* x *w* diagonal matrix of variances at each wavelength. In absorption

spectroscopy, where the precision of measurement is poor in spectral regions where the absorbance is very high (and the light level and signal-to-noise ratio therefore low), it is common to use the transmittance T or its square T^2 as weighting factors.

Inverse Least Squares (ILS) calibration. ILS is a method that can be used to measure the concentration of an analyte in samples in which the spectrum of the analyte in the sample is not known beforehand. Whereas the classical least squares method models the signal at each wavelength as the sum of the concentrations of the analyte times the analytical sensitivity, the inverse least squares methods use the reverse approach and models the analyte concentration c in each sample as the sum of the signals A at each wavelength times calibration coefficients m that express how the concentration of that species is related to the signal at each wavelength:

$$c_{s1} = m_{\lambda 1}A_{s1,\lambda 1} + m_{\lambda 2}A_{s1,\lambda 2} + m_{\lambda 2}A_{s1,\lambda 2} + \ldots \text{ for all } w \text{ wavelengths.}$$

$$c_{s2} = m_{\lambda 1}A_{s2,\lambda 1} + m_{\lambda 2}A_{s2,\lambda 2} + m_{\lambda 2}A_{s2,\lambda 2} + \ldots$$

and so on for all *s* samples. In matrix form

$$\mathbf{C} = \mathbf{AM}$$

where **C** is the *s*-length vector of concentrations of the analyte in the *s* samples, **A** is the *w* x *s* matrix of measured signals at the *w* wavelengths in the *s* samples, and **M** is the *w*-length vector of calibration coefficients.

Now, suppose that you have a set of standard samples that are typical of the type of sample that you wish to be able to measure and which contain a range of analyte concentrations that span the range of concentrations expected to be found in other samples of that type. This will serve as the *calibration set.* You measure the spectrum of each of the samples in this calibration set and put these data into a *w* x *s* matrix of measured signals **A**. You then measure the analyte concentrations in each of the samples by some reliable and independent analytical method and put those data into a *s*-length vector of concentrations **C.** Together these data allow you to calculate the calibration vector **M** by solving the above equation. If the number of samples in the calibration set is greater than the number of wavelengths, the least-squares solution is:

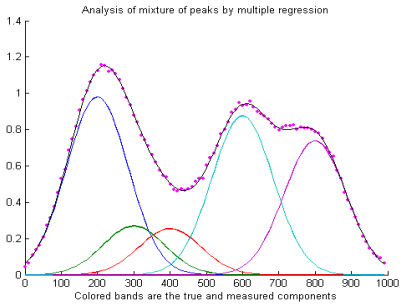
$$\mathbf{M} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{C}$$

(Note that $\mathbf{A}^T\mathbf{A}$ is a square matrix of size *w*, the number of wavelengths, which must be less than *s*). This calibration vector can be used to compute the analyte concentrations of other samples, which are similar to but not in the calibration set, from the measured spectra of the samples:

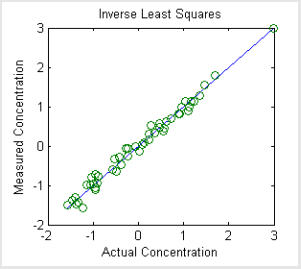
$$\mathbf{C} = \mathbf{AM}$$

Clearly this will work well only if the analytical samples are similar to the calibration set.

Most modern spreadsheets have basic matrix manipulation capabilities and can be used for multicomponent calibration, for example Excel, OpenOffice Calc. Here is an example of a multicomponent calibration calibration performed in a spreadsheet environment. But **Matlab** is really the natural computer approach to multicomponent analysis because it handles all types of matrix math so easily. In Matlab, the notation is a little different: transpose of a matrix **`A`** is **`A'`**, the inverse of **`A`** is **`inv(A)`**, and matrix multiplication is designated by *. Thus the solution to the classical least squares method above is written **`C = inv(E'*E)*E'*A,`** where **`E`** is the rectangular matrix of sensitivities at each wavelength for each component The script RegressionDemo.m demonstrates the classical least squares procedure for a simulated absorption spectrum of a 5-component mixture, illustrated on the left. In this example the dots represent the observed spectrum of the mixture (with noise) and the five colored bands represent the five components in the mixture, whose spectra are known but whose concentrations in the mixture are unknown. The black line represents the "best fit" to the observed spectrum calculated by the



program. In this example the concentrations of the five components are measured to an accuracy of about 1% relative (limited by the noise in the observed spectrum). The Inverse Least Squares (ILS) technique is demonstrated by this script and the graph below. This is a real data set derived from the near infrared (NIR) reflectance spectroscopy of agricultural wheat samples analyzed for protein content. In this example there are 50 calibration samples measured at 6 wavelengths. The samples had already been analyzed by a reliable, but laborious and time-consuming, reference method. The purpose of this calibration is to establish whether near-infrared reflectance spectroscopy, which can be measured much more quickly on wheat paste preparations, correlates to their protein content. These results indicate that it does, at least for this set of 50 wheat samples, and therefore is it likely that near-infrared spectroscopy should do a pretty good job of estimating the protein content of similar unknown samples. The key is that the unknown samples must be similar to the calibrations samples (except for the protein content). However, this is a very common analytical situation in commerce, where large numbers of samples of a similar predictable type must be analyzed.



Curve fitting C: Non-linear Iterative Curve Fitting ("spectral deconvolution" or "peak deconvolution")

The linear least squares curve fitting described in "Curve Fitting A" is simple and fast, but it is limited to situations where the dependent variable can be modeled as a polynomial with linear coefficients. We saw that in some cases a non-linear situation can be converted into a linear one by a coordinate transformation, but this is possible only in some special cases and, in any case, the resulting coordinate transformation of the noise in the data can result in inaccuracies in the parameters measured in this way.

The most general way of fitting any model to a set of data is the iterative method, a kind of "trial and error" procedure in which the parameters of the model are adjusted in a systematic fashion until the equation fits the data as close as required. This sounds like a brute-force approach, and it's true that, in the days before computers, this method was only grudgingly applied. But its great generality, coupled with advances in computer speed and algorithm efficiency, means that iterative methods are more widely used now than ever before.

Iterative methods proceed in the following general way: (1) the operator selects a model for the data; (2) first guesses of all the non-linear parameters are made; (3) a computer program computes the model and compares it to the data set, calculating a fitting error; (4) if the fitting error is greater that the required fitting accuracy, the program systematically changes one or more of the parameters and loops back around to step 3. This continues until the fitting error is less than the specified error. One popular technique for doing this is called the *Nelder-Mead Modified Simplex*. This is essentially a

way of organizing and optimizing the changes in parameters (step 4, above) to shorten the time required to fit the function to the required degree of accuracy. With contemporary personal computers, the entire process typically takes only a fraction of a second.

The main difficulty of the interactive methods is that they sometime fail to converge at an optimum solution in difficult cases. The standard approach to handle this is to restart the algorithm with another set of first guesses. Iterative curve fitting also takes longer than linear regression - with typical modern personal computers, an iterative fit might take fractions of a second where a regression would take fractions of a millisecond. Still, this is fast enough for many purposes.

Note: the term "spectral deconvolution" or "band deconvolution" is often used to refer to this technique.

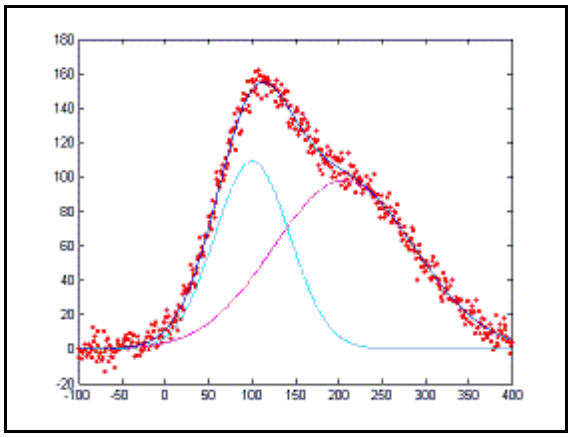
Spreadsheets and stand-alone programs. There are a number of downloadable non-linear iterative curve fitting adds-ons and macros for Excel and OpenOffice, as well as some stand-alone freeware and commercial programs that perform this function.

Matlab has a convenient and efficient function called FMINSEARCH that uses the Nelder-Mead method. It works in conjunction with a user-defined "fitting function" that computes the model, compares it to the data, and returns the fitting error. For example, writing *options = optimset('TolX',0.1);*
`parameter=FMINSEARCH('fitfunction',start,options,x,y)` performs an interative fit of the data in the vectors x,y to a model described in a previously-created function called `fitfunction`, using the first guesses in the vector `start` and stopping at the tolerance defined by the *optimset* function. The parameters of the fit are returned in the vector "parameters", in the same order that they appear in "start".

A simple example is fitting the blackbody equation to the spectrum of an incandescent body for the purpose of estimating its color temperature. In this case there is only one nonlinear parameter, temperature. The script BlackbodyDataFit.m demonstrates the technique, placing the experimentally measured spectrum in the vectors "wavelength" and "radiance" and then calling FMINSEARCH with the fitting function fitblackbody.m.

Another application is demonstrated by Matlab's built-in demo fitdemo.m and its fitting function fitfun.m, which models the sum of two exponential decays. (To see this, just type "fitdemo" in the Matlab command window).

The custom demonstration script Demofitgauss.m demonstrates fitting a Gaussian function to a set of data, using the fitting function fitgauss2.m. In this case there are two non-linear parameters, the peak position and the peak width (the peak height is a linear parameter and is determined by regression in line 9 of the fitting function fitgauss2.m and is returned in the global variable "c"). This is easily extended to fitting two overlapping Gaussians in Demofitgauss2.m (shown on the left) using the *same* fitting function (which easily adapts to any number of peaks, depending on the length of the first-guess "start" vector). All these functions call the user-defined peak shape function gaussian.m. Similar procedures can be defined for other peak shapes simply by calling the corresponding peak shape function, such as lorentzian.m. (Note: in order for scripts like Demofitgauss.m or

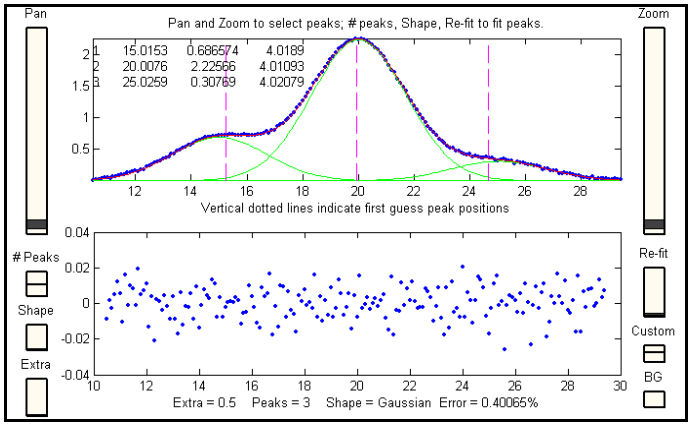


Demofitgauss2.m to work on your version of Matlab, all the functions that they call must be loaded into Matlab beforehand, in this case fitgauss2.m and gaussian.m).

You can create your own fitting functions for any purpose; they are not limited to single algebraic expressions, but can be *arbitrary* complex multi-step algorithms. For example, in *this application* in absorption spectroscopy, a model of the instrumentally-broadened transmission spectrum is fit to the observed transmission data, in order to extend the dynamic range and calibration linearity beyond the normal limits, using a fitting function that performs Fourier convolution of the transmission spectrum with the slit function of the spectrometer. **Note:** you can right-click on any of the m-file links above and select **Save Link As...** to download them to your computer for use within Matlab.

Interactive Peak Fitter

(http://terpconnect.umd.edu/~toh/spectrum/InteractivePeakFitter.htm)



This is a series of Matlab peak fitting programs for time-series signals, which uses an unconstrained non-linear optimization algorithm to decompose a complex, overlapping-peak signal into its component parts. The objective is to determine whether your signal can be represented as the sum of fundamental underlying peaks shapes. These programs do not require the signal processing or optimization toolboxes. They accept signals of any length, including those with non-integer and non-uniform x-values, and can fits groups of peaks with Gaussian, Lorentzian, Logistic, Pearson, and exponentially-broadened Gaussian models (expandable to other shapes). There are three different versions, a command line version, a keypress operated interactive version, and a version with mouse-controlled sliders (which requires Matlab 6.5).

Accuracy and precision of peak parameter measurement

There are four major sources of error in measuring the peak parameters (peak positions, heights, widths, and areas) by iterative curve fitting:

a. Model errors. If you have the wrong model for your peaks, the results can not be

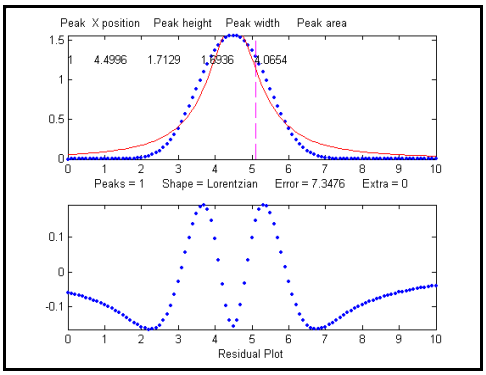
expected to be accurate; for instance, if your actual peaks are Lorentzian in shape, but you fit them with a Gaussian model, or *vice versa*. For example, a single isolated Gaussian peak at x=5, with a height of 1.000 fits a Gaussian model virtually perfectly, using the Matlab user-defined peakfit function:

```
>> x=[0:.1:10];y=exp(-(x-5).^2);
>> [FitResults,MeanFitError]=peakfit([x' y'],5,10,1,1)
FitResults =
1            5            1       1.6649       1.7724
MeanFitError =
  0.001679
```

(The fit results are, from left to right, peak number, peak position, peak height, peak width, and peak area).
But this same peak, when fit with a Logistic model, gives a fitting error of 1.4% and height and width errors or 3% and 6%, respectively:

```
>> [FitResults,MeanFitError]=peakfit([x' y'],5,10,1,3)
FitResults =
1       5.0002
0.96652         1.762
1.7419
MeanFitError =
   1.4095
```

When fit with a Lorentzian model (shown on the right), this peak gives a 6% fitting error and height and width errors of 8% and 20%, respectively.




```
>>
[FitResults,MeanFitError]=peakfit([x' y'],5,10,1,2)
FitResults =
1            5       1.0876       1.3139       2.0579
MeanFitError =
   5.7893
```

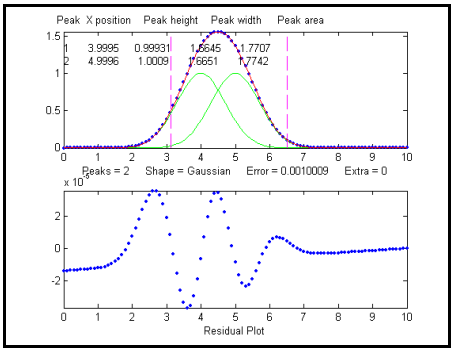
So clearly the larger the fitting errors, the larger are the parameter errors, but the parameter errors are not *equal* to the fitting error (that would just be *too* easy). Also, clearly the peak *width* and *area* are the parameters most susceptible to errors. The peak *positions*, as you can see here, are measured accurately, even if the model is way wrong, as long as the peak is symmetrical and not highly overlapping with other peaks . (To make matters worse, the parameters errors depend not just on the fitting error but also on the data density (number of data points in the width of each peak) and on the extent of peak overlap. It's complicated.)

Another source of model error occurs if you have the wrong number of peaks in your model. For example, if the data actually has two peaks but you try to fit it with only one peak, the peak parameters of the peaks that are fit may not yield accurate parameter measurements. In the example shown on the right, the signal *looks* like one peak, but is actually two peaks at x=4 and x=5 with peaks heights of 1.000 and widths

of 1.665. If you fit this signal with a single-peak model, you get:

```
>> x=[0:.1:10];y=exp(-(x-4).^2)+exp(-(x-5).^2);
>> [FitResults,MeanFitError]=peakfit([x' y'],5,10,1,1)
FitResults =
1          4.5       1.5887       2.1182       3.5823
MeanFitError =
      0.97913
```

But a fit with *two* peaks (shown on the right) is much better and yields accurate parameters for both peaks:

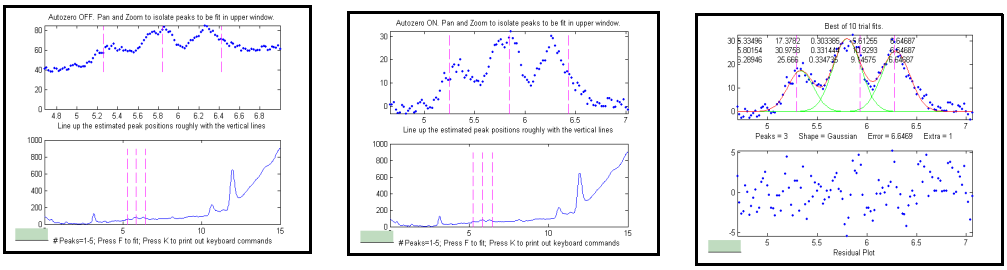


>>

```
[FitResults,MeanFitError]=peakfit([x' y'],5,10,2,1)
FitResults =
1        3.9995       0.99931       1.6645       1.7707
2        4.9996        1.0009       1.6651       1.7742
MeanFitError =
  0.0010009
```

Model errors result in a "wavy" structure in the residual plot (lower panel of the figure), rather than the random scatter of points that would ideally be observed if a peak is accurately fit, save for the random noise. (This is one good reason for not smoothing your data before fitting).

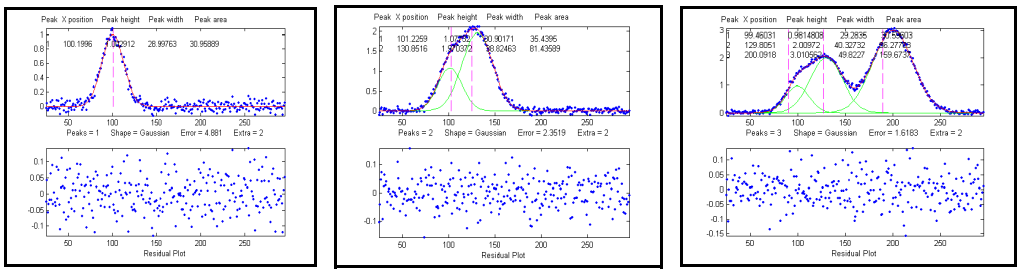
b. Background correction errors. The peaks that are measured in most measurement instruments are often superimposed on a non-specific background. Ordinarily the experiment protocol is designed to minimize the background or to compensate for the background, for example by subtracting the signal from a "blank" from the signal from an actual specimen. But even so there is often a residual background that can not be eliminated completely experimentally. The origin and shape of that background depends on the specific measurement method, but often this background is a broad, tilted, or curved shape, and the peaks are comparatively narrow features superimposed on that background. There are various sophisticated methods described in the literature for estimating and subtracting the background in such cases. The simplest assumption, which is used by the Interactive Peak Fitter, is that the background is locally linear, that is, can be approximated as a straight line in the local region of group of peaks being fit together. When the autozero mode of the ipf.m function is turned on (**T** key), a straight-line baseline connecting the two ends of the signal segment in the upper panel will be automatically subtracted as the pan and zoom controls are used to isolate the group of overlapping peaks to be fit.



Example of an experimental chromatographic signal. From left to right, (1) Raw data with peaks superimposed on baseline; (2) Baseline automatically subtracted by the autozero mode in ipf.m; (3) Fit with a three-peak Gaussian model.

c. Random noise in the signal. Any experimental signal has a certain amount of random noise, which means that the individual data points scatter randomly above and below their mean values. The assumption is ordinarily made that the scatter is equally above and below the true signal, so that the long-term average approaches the true mean value; the noise "averages to zero" as it is often said. The practical problem is that any given recording of the signal contains only one finite sample of the noise. If another recording of the signal is made, it will contain another independent sample of the noise. These noise sample are not infinitely long and therefore do not represent the true long-term nature of the noise. This presents two problems: (1) an individual sample of the noise will not "average to zero" and thus the parameters of the best-fit model will not necessarily equal the true values, and (2) the magnitude of the noise during one sample might not be typical; the noise might have been randomly greater or smaller than average during that time. This means that the mathematical "propagation of error" methods, which seek to estimate the likely error in the model parameters based on the noise in the signal, will be subject to error (*underestimating* the error if the noise happens to be *lower* than average and *overestimating* the errors if the noise happens to be *larger* than average).

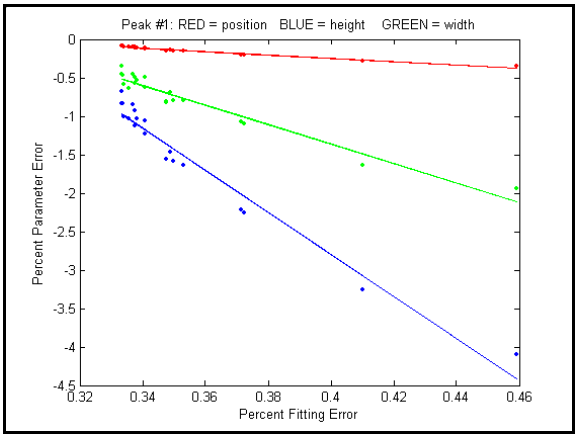
A better way to estimate the parameter errors is to record multiple samples of the signal, fit each of those separately, compute the models parameters from each fit, and calculate the standard error of each parameter. This is exactly what the script *DemoPeakfit.m* (which requires the peakfit.m function) does for simulated noisy peak signals such as those illustrated below. It's easy to demonstrate that, as expected, the average fitting error precision and the relative standard deviation of the parameters increases directly with the random noise level in the signal. But the precision and the accuracy of the measured parameters *also* depend on which parameter it is (peak positions are always measured more accurately than their heights, widths, and areas) and on the peak height and extent of peak overlap (the two left-most peaks in this example are not only weaker but also more overlapped that the right-most peak, and therefore exhibit poorer parameter measurements). In this example, the fitting error is 1.6% and the percent relative standard deviation of the parameters ranges from 0.05% for the peak position of the largest peak to 12% for the peak area of the smallest peak.



The parameter errors depend not only on the characteristics of the peaks in question, but also upon other peaks that are overlapping it. From left to right: (1) a single peak at x=100 with a peak height of 1.0 and width of 30 is fit with a Gaussian model, yielding a relative fit error of 4.9% and relative standard deviation of peak position, height, and width of 0.2%, 0.95%, and 1.5% , respectively. (2) The same peak, with the same noise level but with another peak overlapping it, reduces the relative fit error to 2.4% (because the addition if the second peak increases overall signal amplitude), but increases the relative standard deviation of peak position, height, and width to 0.84%, 5%, and 4% - a seemingly better fit, but with poorer precision for the first peak. (3) The addition of a third peak further reduces the fit error to 1.6% , but the relative standard deviation of peak position, height, and width of the first peak are still 0.8%, 5.8%, and 3.64%, about the same as with two peaks, because the third peak does not overlap the first one significantly.

One way to reduce the effect of noise is to take more data. If the experiment makes it possible to reduce the x-axis interval between points, or to take multiple readings at each x-axis values, then the resulting increase in the number of data points in each peak should help reduce the effect of noise. As a demonstration, using the script DemoPeakfit.m to create a simulated overlapping peak signal like that shown above right, it's possible to change the interval between x values and thus the total number of data points in the signal. With a noise level of 1% and 75 points in the signal, the fitting error is 0.35 and the average parameter error is 0.8%. With 300 points in the signal and the same noise level, the fitting error is essentially the same, but the average parameter error drops to 0.4%, suggesting that the accuracy of the measured parameters varies inversely with the square root of the number of data points in the peaks.

d. Iterative fitting errors. Unlike multiple linear regression curve fitting, iterative methods may not converge on the exact same model parameters each time the fit is repeated with slightly different starting values (first guesses). The Interactive Peak Fitter makes it easy to test this, because it uses slightly different starting values each time the signal is fit (by pressing the **F** key in ipf.m, for example). Even better, by pressing the **X** key, the ipf.m function



silently computes 10 fits with different starting values and takes the one with the lowest fitting error. A basic assumption of any curve fitting operation is that the fitting error (the RMS difference between the model and the data) is minimized, the parameter errors (the difference between the actual parameters and the parameters of the best-fit model) will also be minimized. This is generally a good assumption, as demonstrated by the graph to the right, which shows typical percent parameters errors as a function of fitting error for the left-most peak in one sample of the simulated signal generated by DemoPeakfit.m (shown in the previous section). The variability of the fitting error here is caused by random small variations in the first guesses, rather than by random noise in the signal. In many practical cases there is enough random noise in the signals that the iterative fitting errors within one sample of the signal are small compared to the random noise errors between samples.

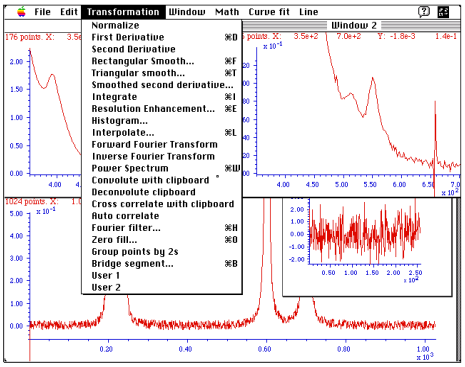
Remember that the variability in measured peak parameters from fit to fit of a single sample of the signal is *not* a good estimate of the precision or accuracy of those parameters, for the simple reason that those results represent only one sample of the signal, noise, and background. The sample-to-sample variations are likely to be much greater than the within-sample variations due to the iterative curve fitting. (In this case, a "sample" is a single recording of signal).

So, to sum up, we can make the following observations about the accuracy of model parameters: (1) the parameter errors are directly proportional to the noise in the data and to the fitting error (but is not equal to the fitting error); (2) the errors are typically least for peak position and worse for peak width and area; (3) the errors depend on the data density (number of independent data points in the width of each peak) and on the extent of peak overlap (the parameters of isolated peaks are easier to measure than highly overlapped peaks).

Appendix: Software details

1. SPECTRUM

Many of the figures in this essay are screen images from S.P.E.C.T.R.U.M. (**S**ignal **P**rocessing for **E**xperimental **C**hemistry **T**eaching and **R**esearch/ **U**niversity of **M**aryland), a Macintosh program we have developed for teaching signal processing to chemistry students.



SPECTRUM is designed for post-run (rather than real-time) processing of "spectral" or time- series data (y values at equally-spaced x intervals), such as spectra, chromatograms, electrochemical signals, etc. The program enhances the information content of instrument signals, for example by reducing noise, improving resolution, compensating for instrumental artifacts, testing hypotheses, and decomposing a complex signal into its component parts.

*SPECTRUM was the winner of two EDUCOM/NCRIPTAL national software awards in 1990, for **Best Chemistry** software and for **Best Design**.*

Features

- Reads one- or two- column (y-only or x-y) text data tables with either tab or space separators
- Displays fast, labeled plots in standard resizable windows with full x- and y-axis scale expansion and a mouse- controlled measurement cursor
- Addition, subtraction, multiplication, and division of two signals
- Two kinds of smoothing.
- Three kinds of differentiation
- Integration
- Resolution enhancement
- Interpolation
- Fused peak area measurement by perpendicular drop or tangent skim methods, with mouse-controlled setting of start and stop points
- Fourier transformation
- Power spectra
- Fourier filtering
- Convolution and deconvolution
- Cross- and auto-correlation
- Built-in signal simulator with Gaussian and Lorentzian bands, sine wave and normally-distributed random noise
- A number of other useful functions, including: inspect and edit individual data points, normalize, histogram, interpolate, zero fill, group points by 2s, bridge segment, superimpose, extract subset of points, concatenate, reverse X-axis, rotate, set X axis values, reciprocal, log, ln, antilog, antiln, standard deviation, absolute value, square root

SPECTRUM can be used both as a research tool and as an instructional aid in teaching signal processing techniques. The program and its associated tutorial was originally developed for students of analytical chemistry, but the program could be used in any field in which instrumental measurements are used: e.g. chemistry, biochemistry, physics, engineering, medical research, clinical psychology, biology, environmental and earth sciences, agricultural sciences, or materials testing.

Machine Requirements: Any Macintosh model with minimum 1 MByte RAM, any standard printer. Color screen desirable. SPECTRUM has been tested on most Macintosh models and on all versions of the operating system through OS 8.1.

PC users can run SPECTRUM using a Macintosh emulator program running on a

Windows machine. Currently available Macintosh emulators include SoftMac(http://www.emulators.com/download.htm), Basilisk II (http://basilisk2.cjb.net/), Basilisk II JIT (http://gwenole.beauchesne.online.fr/basilisk2/), and vMac (http://www.vmac.org/).

The full version of SPECTRUM 1.1 is now available as freeware, and can be downloaded from http://terpconnect.umd.edu/~toh/spectrum/. There are two versions:

> **SPECTRUM 1.1e**: Signals are stored internally as *extended-precision* real variables and there is a limit of 1024 points per signal. This version performs all its calculations in extended precision and thus has the best dynamic range and the smallest numeric round-off errors. The download address of this version in HQX format is http://terpconnect.umd.edu/~toh/spectrum/SPECTRUM11e.hqx.
>
> **SPECTRUM 1.1b**: Signals are stored internally as *single-precision* real variables and there is a limit of 4000 points per signal. This version is less precise in its calculations (has more numerical round-off error) than the other version, but allows signals with data more points. The download address of this version in HQX format is http://terpconnect.umd.edu/~toh/spectrum/SPECTRUM11b.hqx.

The two versions are otherwise identical.

There is also a documentation package (located at http://terpconnect.umd.edu/~toh/spectrum/SPECTRUMdemo.hqx) consisting of:

> **a. Reference manual.** Macwrite format (Can be opened from within MacWrite, Microsoft Word, ClarisWorks, WriteNow, and most other full-featured Macintosh word processors). Explains each menu selection and describes the algorithms and mathematical formulae for each operation. The SPECTRUM Reference Manual is also available separately in PDF format at http://terpconnect.umd.edu/~toh/spectrum/SPECTRUMReferenceManual.pdf.
>
> **b. Signal processing tutorial.** Macwrite format (Can be opened from within MacWrite, Microsoft Word, ClarisWorks, WriteNow, and most other full-featured Macintosh word processors). Self-guided tutorial on the applications of signal processing in analytical chemistry. This tutorial is also available on the Web at (http://terpconnect.umd.edu/~toh/Chem498C/SignalProcessing.html)
>
> **c. Tutorial signals:** A library of prerecorded data files for use with the signal processing tutorial. These are plain decimal ascii (tab-delimited) data files.

These files are binhex encoded: use Stuffit Expander to decode and decompress as usual. If you are downloading on a Macintosh, all this should happen completely automatically. If you are downloading on a Windows PC, shift-click on the download links above to begin the download. If you are using the ARDI Executor Mac simulator, download the "HQX" files to your C drive, launch Executor, then open the downloaded HQX files with Stuffit Expander, which is pre-loaded into the Executor Macintosh environment. Stuffit Expander will automatically decode and decompress the downloaded files. Note: Because it was developed for academic teaching application where the most modern and powerful models of computers may not be available, SPECTRUM was designed to be "lean and mean" - that is, it has a simple Macintosh-type user interface and very small memory and

disk space requirements. It will work quite well on Macintosh models as old as the Macintosh II, and will even run on older monochrome models (with some cramping of screen space). It does not require a math co-processor.

(c) 1989 T. C. O'Haver. This program is free and may be freely distributed. It may be included on CD-ROM collections or other archives.

2. Matlab

Matlab is a high-performance commercial numerical computing environment and programming language that is widely used in research and education. See http://en.wikipedia.org/wiki/MATLAB for a general description. There are several basic on-line tutorials and collection of sample code. For example:

a. MATLAB Tutorial for New Users (http://www.youtube.com/watch?v=MdrShPzHeYg). This is a narrated 4-minute video introduction for new users.

b. An Introductory Guide to MATLAB. (http://www.cs.ubc.ca/spider/cavers/MatlabGuide/guide.html)

c. Matlab Summary and Tutorial. (http://www.math.ufl.edu/help/matlab-tutorial/)

d. A Practical Introduction to Matlab (http://www.math.mtu.edu/~msgocken/intro/intro.html)

e. Matlab Chemometrics Index http://www.mathworks.com/matlabcentral/link_exchange/MATLAB/Chemometrics/index.html

f. Multivariate Curve Resolution. http://www.ub.edu/mcr/welcome.html

---

References

1. Douglas A. Skoog, *Principles of Instrumental Analysis*, Third Edition, Saunders, Philadelphia, 1984. Pages 73-76.

2. Gary D. Christian and James E. O'Reilly, *Instrumental Analysis*, Second Edition, Allyn and Bacon, Boston, 1986. Pages 846-851.

3. Howard V. Malmstadt, Christie G. Enke, and Gary Horlick, *Electronic Measurements for Scientists*, W. A. Benjamin, Menlo Park, 1974. Pages 816-870.

4. Stephen C. Gates and Jordan Becker, *Laboratory Automation using the IBM PC*, Prentice Hall, Englewood Cliffs, NJ, 1989.

5. Muhammad A. Sharaf, Deborah L Illman, and Bruce R. Kowalski, *Chemometrics*, John Wiley and Sons, New York, 1986.

6. Peter D. Wentzell and Christopher D. Brown, Signal Processing in Analytical Chemistry, in *Encyclopedia of Analytical Chemistry*, R.A. Meyers (Ed.), p. 9764–9800,

John Wiley & Sons Ltd, Chichester, 2000 (http://myweb.dal.ca/pdwentze/papers/c2.pdf)

7. Constantinos E. Efstathiou, Educational Applets in Analytical Chemistry, Signal Processing, and Chemometrics. (http://www.chem.uoa.gr/Applets/Applet_Index2.htm)

8. A. Felinger, Data Analysis and Signal Processing in Chromatography, Elsevier Scice (19 May 1998).

9. Matthias Otto, Chemometrics: Statistics and Computer Application in Analytical Chemistry, Wiley-VCH (March 19, 1999). Some parts viewable in Google Books.

10. Steven W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing. (Downloadable chapter by chapter in PDF format from http://www.dspguide.com/pdfbook.htm). This is a much more general treatment of the topic.

11. Robert de Levie, *How to use Excel in Analytical Chemistry and in General Scientific Data Analysis*, Cambridge University Press; 1 edition (February 15, 2001), **ISBN-10:** 0521644844. PDF excerpt .

12. Scott Van Bramer, Statistics for Analytical Chemistry, http://science.widener.edu/svb/stats/stats.html.

13. Numerical Analysis for Chemical Engineers, Taechul Lee (http://www.cheric.org/ippage/e/ipdata/2001/13/lecture.html)

14. Educational Matlab GUIs, Center for Signal and Image Processing (CSIP), Georgia Institute of Technology. (http://users.ece.gatech.edu/mcclella/matlabGUIs/)

15. Digital Signal Processing Demonstrations in Matlab, Jan Allebach, Charles Bouman, and Michael Zoltowski, Purdue University (http://www.ecn.purdue.edu/VISE/ee438/demos/Demos.html)