RAZOR LIBRARY

Resolution Enhancement,
Smoothing, Derivatives,
Peak Picking, Peak Fitting,
and Baseline Estimation

using Bayesian,
Maximum Likelihood,
and Maximum Entropy
Spectral Analysis Methods

Version C4.0

Quick Start

- Copy all the Razor Library files into the desired directory on the destination hard drive.
- Select the chapter of this manual which describes the Maximum Likelihood, Maximum Entropy, or Bayesian method you wish to use. The calls for the principal routines are found on the following pages:

- Study the annotated source code supplied in handle.c for an example of how to implement the desired algorithm.
- Technical Support:

Dr. Lin DeNoyer Ph: 607-272-6735 Email: lkd1@cornell.edu
Dr. Jack Dodd Ph: 607-847-6944 Email: jackdodd@clarityconnect.com

Changes for Vers 4.0

(released May 1998)

- A new baseline algorithm **rzredg** has been added.
- **rzrfit** requires larger arrays datmat and work. These changes have been made to prepare for input limits on the fitted parameters.
- **rzrfit** convergence has been improved.

Changes for Vers 3.2

(released May 1997)

- **rzrfit** allows user to specify scale factor for Poisson-noise data. See new instructions for vnoise on p. 98.
- The (scaled) entropy for **rzrdec** has been given the correct (negative) sign.

Changes for Vers 3.1

(released July 1996)

- A new function **rzrstr**, which is a linearized form of the ***classic equation for Maximum Entropy*** **deconvolution** (p. 68), is described on p. 73. This function provides a superior alternative to Fourier deconvolution.
- The mathematical foundations of **rzrdec** are fully described in the manual. (See p. 68). **rzrdec** IS classic Maximum Entropy deconvolution.
- The functions **rzrdec**, **rzrash**, **rzrluc**, **rzrstr**, and **rzrfit** allow the user to either input a noise value, or request auto-calculation of the noise in the input data. A new method for calculating the noise in the data gives better results in low-noise cases.
- **rzrfit** will fit a model to selected regions within a file. See new instructions for vnoise on p. 98.

Changes for Vers 3.0

(released February 1996)

- **rzrdec** has been improved. Entropy of the resolution-enhanced configuration is calculated each iteration. The input parameters for this function have changed.

- **rzrfit** has new capabilities. Peaks may be 'linked' to each other in a master/slave relationship. A single master peak may be linked to any number of slave peaks through position offsets, height ratios, width ratios, or other parameter ratios. Linking peaks in this manner is especially valuable for x-ray spectroscopy. The Pearson7 peakshape now comes in both symmetric and asymmetric varieties. The input parameters for **rzrfit** have changed.
- The **rzrfit** engine has been fortified for heavy-duty work. This peak-fitting engine will converge under harsh conditions which cause others to fail.
- All integers are now declared as *long* (4-byte) integers, which allows processing of longer data arrays, and helps maintain uniformity for compilation with many different compilers.

Changes for Versions 2.0 - 2.6

- **rzrfit** will automatically process all the peaks in a large data array. It will automatically identify a peak 'bunch', and process those peaks together, then move on to identify and process the next bunch. This bunch-mode of processing is a lot *faster* than the all-at-once method!
- **rzrfit** contains two new analytic peakshapes, Pearson VII and Log Normal.
- **rzrfit** automatically checks itself for convergence, and tells you when it is finished.
- When you set up a peakshape by identifying an isolated peak from a data file or a data array, you usually need to remove a baseline, and often need to smooth, the *real data* peakshape. A new utility **rzrcpk** (extract peak) performs these functions with ease. (See Page 179 and **handle.for**).
- A new utility **rzpkst** will resort the peak arrays filled by **rzrpic** and **rzrbas**. The peaks may be sorted by significance, height, width, or location. **rzpkst** is described on Page 175, and source code for using this utility is given in **handle.for**.
- A new utility **rzdfil** will help you fill the **datmat** input array for **rzrfit**, using the output arrays from **rzrpic** and **rzrbas**. See Page 177, and **handle.for**.
- **rzrpic** and **rzrbas** will automatically search for *negative peaks*, if a negative peak is presented in the **shape** array.

Contents

Chapter 1

Razor Library Description

Razor Library is furnished as an object code library. All versions **assume the presence of a numeric coprocessor**.

The Razor Library contains:

- An object code library, RZRxxxx.LIB. Royalties apply for commercial distribution of programs containing Razor Library object code.
- Source code is provided for many functions, in the file **rzrserve.c**. You are free to modify any source code for your own use.
- A simple handling program, **handle.c**, is provided in source, and as an executable program, to illustrate the calls and necessary input for each of the principal functions.
- Sample data files are included to illustrate Razor's capabilities.

The C Razor Library is written in almost ANSI C, in order to be compatible with as many compilers as possible. Object code for other C and Fortran compilers, and for other operating systems, is available. Call Spectrum Square Associates, 607-272-2352, for information.

1.1 Advanced Statistical Functions

The core of Razor Library consists of fourteen principal functions. Twelve of these functions are based upon Maximum Likelihood and/or Maximum Entropy principles. They have already proven useful in both spectroscopy and chromatography, for smoothing (RazorSmooth), enhancing resolution (RazorSharp), peak fitting (RazorFit), peak picking (RazorPick), reducing noise in ratio spectra (RazorDivide), and baseline removal (RazorBase). The functions are quite general, and may be used on any linear data array where the data have been sampled in equispaced intervals.

All of the RazorSmooth, RazorPick, RazorFit, RazorSharp, RazorDerivative, RazorDivide, and RazorNoise statistical functions are based on Maximum Likelihood/Maximum Entropy and Bayesian principles. They were developed and/or programmed by PhD physicists at Spectrum Square Associates. The mathematical equations behind these statistical methods are given in the appropriate chapters of this manual.

At present, the only other implementation of these powerful methods are the PC-based products **RAZOR SR.**, **SQUARE TOOLS**, and **RAZOR for GRAMS/386**, also developed at Spectrum Square. **RAZOR SR.** is a complete spectral data processing program with extensive batch capabilities and additional functions specifically tailored for diode arrays and for micro-Raman analysis. **SQUARE TOOLS** and **RAZOR for GRAMS/386** are sets of addon programs for Galactic Industries' data analysis programs Spectra Calc^{tm}, Lab Calc^{tm}, and GRAMS/386^{tm}.

1.2 Principal Razor functions

Fourteen principal functions are described in subsequent chapters of this manual. The programmer has access to all of the Maximum Likelihood and Maximum Entropy capabilites of Razor Library through these functions. Razor capabilities and its fourteen principal functions are:

RazorSmooth: Maximum Likelihood estimation of the smooth parent distribution of a noisy data set.
RazorEntropySmooth — **rzresm**
RazorPoissonSmooth — **rzrpsm**
RazorNormalSmooth — **rzrnsm**

RazorDivide: Maximum Likelihood smoothing of the ratio of two noisy data sets, such as smoothing sample/reference spectra.
— **rzrdiv**

RazorSharp: Maximum Likelihood and Maximum Entropy/Bayesian resolution sharpening and enhancement.
RazorA-Sharp — **rzrash**
RazorDeconvolve — **rzrdec**
RazorLucy — **rzrluc**

RazorDerivative: Bayesian Derivatives.
— **rzrdif**

RazorPick: Maximum Likelihood/Bayesian peak picking.
— **rzrpic**

RazorFit: Maximum Likelihood fitting model peaks to data.
— **rzrfit**

RazorBaseline: Maximum Likelihood and other methods for estimating the baseline of a data set.

RazorBase — **rzrbas**
RazorQuickBase — **rzrqba**
RazorEdge — **rzredg**
RazorCut — **rzrcut**

RazorNoise: Maximum Likelihood estimation of the noise vector of a data set.

— **rzrnoi**

The fourteen principal statistical functions of the Razor Library are provided only as object code.

1.3 User input and programmer control

The fourteen principal functions require additional user input besides the data array. This input usually takes two forms: **knowledge about the type of noise** present in the data, and **knowledge of the intrinsic shapes of peaks** in the data.

In this manual, the required input for each algorithm is emphasized at the beginning of the chapter which describes the algorithm. Often, such input must be based upon measurements derived from an observed spectrum. The mechanism for obtaining the user input is the responsibility of the programmer.

Programmer notes are given for many of the functions, describing shortcuts, ways to save space, or other technical aspects of the functions.

Some of the functions are iterative. Iterations are always under the control of the programmer. The programmer notes describe appropriate convergence criteria, or tell the programmer when to quit. The programmer always has the option of displaying intermediate results for the user, if he wishes. Every effort has been made to avoid the "black box" syndrome, by making as many parameters as possible accessible to the programmer.

1.4 Example source code

Most programmers will want to get these routines up and running as rapidly as possible. We have provided a demonstration program called **handle** for that purpose. Handle is a very simple example of how input and output may be implemented. The file **handle.c** contains documented source code which you are free to use, or modify for incorporation into your own data processing system. (**Handle** contains no graphical interface, however.)

1.5 Source for service functions

Source code is provided for all service functions, in the file **rzrserve.c**. A discussion of those routines which you may wish to change, and the circumstances under which you might wish to change them, is given in Chapter 11.

The most important service functions you should be aware of are those which generate analytical peak shapes. The RazorFit algorithm requires explicit analytical peak shapes, and Razor Library contains a set of functions which are of the proper format, and which are called by RazorFit. You may add additional peak shapes as your needs demand. See the source listing for instructions. Many of the other principal functions also require peak shapes. You may wish to use the shapes functions to generate peak shapes in memory, whenever appropriate.

Chapter 2

RazorSmooth — rzresm/rzrpsm/rzrnsm

2.1 Smoothing which Preserves Resolution

RazorSmooth is set of Maximum Likelihood and Maximum Entropy smoothing functions for many types of noise problems. The functions **estimate the smoothed data set that would be achieved if the user could average many, many scans** similar to the one at hand. Such a smoothed data set is usually called the parent distribution. The theory is described in 'Maximum Likelihood smoothing of noisy data,' published in American Laboratory, March 1990, in International Laboratory, June 1990, and in later sections of this chapter. The functions provide the maximum amount of smoothing possible, consistent with minimum loss of resolution in the displayed data.

The **RazorSmooth** functions give:

- Optimum smoothing for the declared noise statistics.
- Almost no loss of resolution when peakshapes are accurately known. (Clearly, there would be **no** loss of resolution if one could obtain the true parent distribution. However, the *estimated* parent distribution never achieves the ideal.)

2.2 Which one to use?

The programmer (or user) must decide whether the noise statistics are closer to a Normal distribution or a Poisson distribution.

Razor Poisson Smooth (rzrpsm), for Poisson noise, is an iterative solution which requires considerably more time. It constrains the smoothed solution to be positive. (Page 14.)

Razor Entropy Smooth (rzresm) is a fast, excellent approximation to the full Maximum Likelihood solution for Normally- distributed noise. (Page 8.)

Razor Normal Smooth (rzrnsm), for Normal noise, is an iterative solution which requires considerably more time. It constrains the smoothed solution to be positive. (Page 20.)

2.3 rzresm — Razor Entropy Smooth

Razor Entropy Smooth provides a Maximum Likelihood estimate of a noise-free parent spectrum, where the observed spectrum is a single noisy example drawn from this parent. The noise is assumed to come from a Normal distribution.

rzresm is a fast, excellent approximation to the full Maximum Entropy solution for Normally-distributed noise. It is also a Bayesian method. (Section 2.12. Page 26.)

The required user input for **rzresm** is:

- Data array.
- Peakshapes - either true or estimated. It is not critical that the user choose an exact peakshape for **rzresm**. When all the peaks in the data are not the same, the user should select a smooth peakshape characteristic of the *narrowest* feature of interest in the data. Do **not** choose a peakshape that is too wide, else you *will* obtain false results.

Processing notes:

- The estimated parent spectrum is not constrained to be positive in this solution.

Programmer notes:

- **rzresm** requires 3 full-sized arrays, **ydata, yout**, and **trans**.

  The number of arrays may be reduced to 2 by setting the output array **yout** to the input array **ydata**.
- **ydata** *will not* be altered outside the data region **0** - **n2**, unless you elect to do the processing in-place by setting **yout** = **ydata**.
- If you are processing many scans, all of the same length, and using the same peakshape for all, save processing time with this tactic. Call **rzresm** the first time with **newpk** = 1, thereafter with **newpk** unchanged. When **newpk** = 1, all the functions in Razor Library transfer a properly scaled, properly phased, copy of the input **shape** array into the array **trans**, and then perform an FFT on the **trans** array. When **newpk** $>$ 1, the functions ignore **shape**, and use **trans** directly. This saves the time of a Fourier transform on **trans**. (Note that when **newpk** $>$ 1, **shape** can be a dummy array of length 1, since it will not be used.)

long rzresm(float ydata[ ], long n2, float shape[ ], long nl2,
float yout[ ], float trans[ ], long *n, long *newpk,
long *nfwhm, double *sigma)

Input arrays which must be filled:
  ydata, filled between **0** and **n2**, length **n2+1**
  shape, filled between **0** and **nl2**
    NOTE: **shape** will be read only, not altered.
    NOTE: If **newpk** > 1, **shape** will not be read.
Additional arrays to be furnished:
  yout, length **n**
  trans, length **n**
Input variables: **n2, nl2, n, newpk**
  n2 is the index of the last data value in **ydata**
  nl2 is the index of the last data value in **shape**
  n is the size of arrays **ydata, yout** and **trans**
  newpk indicates whether or not **shape** is a new peakshape
Output arrays:
  yout, filled between **0** and **n2**
Output variables:
  n = amount of array space used
    NOTE: if **n** is returned negative, abs(n) = amount of array space needed (but not available). Operation not successful.
  newpk = n if **trans** was successfully loaded from **shape**
  nfwhm = full-width-at-half-maximum of peakshape
  sigma = RMS noise in **ydata**
Function return values:
  rzresm = 0 if successful
    If **rzresm** < 0, error occurred
    Use **rzrerr** (page 174) to obtain error text

Description of variables

ydata on *input* is the *raw data array*. It should contain the raw data between data points **0** and **n2**. **ydata** will not be altered outside this range.

ydata must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

n2 is the *last location* of data in the **ydata** array. **n2** is to be furnished as *input*.

shape is an *input* array which holds the *peakshape of the narrowest spectral feature* in **ydata** which is of interest to the user. The relevant peakshape is located between data points **0** and **nl2** in **shape**.

nl2 is *input* and the *index of the last data point of the peakshape* in **shape**. We recommend that **nl2**+1 be at least 6⋆**nfwhm**, and that the peak be approximately centered in the (**0**,**nl2**) interval.

yout is the *output smoothed data array*. It will be smoothed between data points **0** and **n2**, and should be ignored outside this range.

yout must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

trans is an *array of size n* which will be used to house the Fourier transform of the peakshape. The amount of space used in **trans** is calculated in **rzprep**. See the discussion below for **n**.

trans is either empty or filled, depending on the parameter **newpk**. Whenever **newpk** = 1, it is assumed that the contents of **shape** have been altered, and **trans** is properly loaded by **rzresm**. When **newpk** > 1, it is expected that **trans** has not been changed since the last time it was filled. See the discussion below for **newpk**.

n is *input* as the *amount of space furnished* in the **yout**, and **trans** arrays.

The function **rzsizn** will calculate **n**, the minimum amount of space needed. The required size of **n** is determined by **n2** and by the width of the peak in the **shape** array. Obtain the minimum required **n** with this call:

n = rzsizn(n2,shape,nl2)

On *output*, **n** is the amount of *space used for the Fourier transforms* in the **yout**, and **trans** arrays. If **n** is negative on output, the amount of space furnished was inadequate, and no processing has taken place. If **n** is returned negative, then abs(**n**) is the amount of space needed in the above arrays.

The space required for the Fourier transform is always calculated in **rzprep**, described in Chapter 11. When **newpk** = 1, **rzprep** calculates the required size of the Fourier transform as the smallest power of two larger than (**n2**+1+3***nfwhm**). You may wish to calculate **n** in an alternate fashion. See Chapter 11.

NOTE: When **rzresm** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you with to process additional data with the same peakshape, you need not change either **newpk** or **n**, provided that (a) your peakshapes do not change, and (b) your input **ydata** sizes (**n2**+1) do not increase.

newpk on *input* is an *integer flag* set which should be initially set to 1. It informs the peakshape processor that a new peakshape is present in **shape**. The processor measures certain parameters of the new peakshape, and then fills the **trans** array with the Fourier transform of a properly shifted and scaled peakshape. When the peakshape processor finishes successfully, it will *output* **newpk** = **n**, where **n** is the actual space used in **trans**.

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1.* Whenever **rzresm** is called with **newpk** > 1, ensure that:

(a) The user wants to use the previous peakshape for the current processing, and **trans** is not changed.

(b) The size of the array needed to transform the new data set is no larger than the **n** used previously. If this second criterium is violated, the *output* value of **rzresm** will be **rzresm** = -2.

nfwhm is *output* as the number of *data points between the half-maxima* of the peakshape feature in **shape**.

sigma is *output* as the *standard deviation (root- mean-square) of the noise* which has been removed by the smoothing process.

2.4 Example using rzresm

Raman microprobe spectra rarely have enough photons. SPEC6, which is a microprobe spectrum of a carbon thin film, is no exception. We smoothed the spectrum shown below using a Lorentzian, 150 points wide, stored in PEAK6.

Data file: SPEC6

Peakshape file: PEAK6

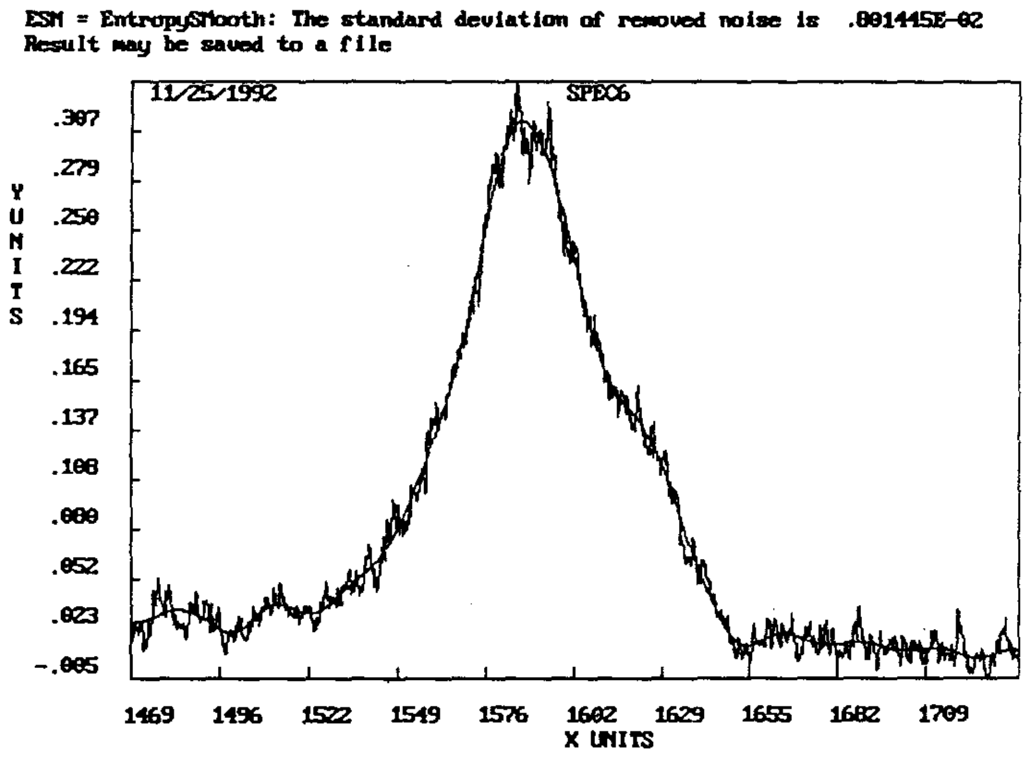
Using HANDLE:

```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
ESM=EntropySMooth. Smooths Normal (thermal/gaussian) noise. ME
PSM=PoissonSMooth. Smooths Poisson (counting) noise. ML.
NSM=NormalSMooth. Smooths Normal noise. ML.
DIV=RazorDivide. Calculates transmission spectra. ML.
ASH=RazorASharp. Enhances resolution. ML.
DEC=RazorDeconvolve. Maximum Entropy deconvolution. ME/Bayesian.
LUC=RazorLucy. Classic ML deconvolution.  ML.
DIF=RazorDerivative. Derivatives 0th-nth. Bayesian.
PIC=RazorPick. Finds peak positions for FIT. ML/Bayesian.
FIT=RazorFit. Fits model peaks to data.  ML.
BAS=RazorBase. Finds baseline. ME/Bayesian.
QBA=RazorQuickBase. Finds baseline.
EDG=RazorEdge. Fits baseline to lower edge of data.
NOI=RazorNoise. Finds noise spectrum. ML.
GEN=Generates synthetic peakshape.
SAV=Save result, QUI=Quit.
Choose an operation (3 uppercase characters): ESM

Enter name of spectrum: SPEC6
Enter name of peakshape: PEAK6

Enter RZRESM. Please wait for processing...
The RMS noise is 0.00711781
The FWHM of the peakshape is 149
The size of the array space used was 2048
RESULT MAY BE SAVED TO A FILE

Press ENTER to return to menu.
```



```
ESM = EntropySMooth: The standard deviation of removed noise is  .891445E-02
Result may be saved to a file
```

2.5 rzrpsm — Razor Poisson Smooth

Razor Poisson Smooth (**rzrpsm**) provides a Maximum Likelihood estimate of a noise-free parent spectrum. The observed spectrum is a single a noisy example drawn from this parent spectrum. The noise is assumed to come from a Poisson distribution.

The required user input for **rzrpsm** is:

- Data array. *The input data set must be positive,* as is appropriate for data with Poisson noise. **It is the user's responsibility to remove the correct offset from the raw data before using rzrpsm.**
- Peakshapes - either true or estimated. It is not critical that the user choose an exact peakshape for **rzrpsm**. When all the peaks in the data are not the same, the user should select a smooth peakshape characteristic of the *narrowest* feature of interest in the data.

Processing notes:

- The function produces an estimated parent spectrum which is constrained to be positive.
- This function is iterative, and therefore takes considerably more time than **rzresm**.

Programmer notes:

- **rzrpsm** requires 4 full-sized arrays. If space is a problem, see Section 11.2.
- Set iter = 0 for the initial call. **rzrpsm** will then maintain iter for you. **rzrpsm** needs 15 to 25 iterations. Some peakshapes converge faster. When the peakshape is Gaussian, the convergence is faster than when the peakshape is Lorentzian.

long rzrpsm(float ydata[ ], long n2, float shape[ ], long nl2,
float yout[ ], float w[], float trans[ ], long *n, long *newpk,
long *iter, long *nfwhm, double *sigma)

Input arrays which must be filled:
- **ydata**, filled between **0** and **n2**, length **n**
- **shape**, filled between **0** and **nl2**
  - NOTE: If **newpk** = 0, **shape** will not be used.
  - NOTE: **shape** will be read only, not altered.

Additional arrays to be furnished:
- **yout**, length **n**
- **w**, length **n**
- **trans**, length **n**

Input variables: **n2, nl2, n, newpk**
- **n2** is the index of the last data values in **ydata**
- **nl2** is the index of the last data value in **shape**
- **n** is the size of arrays **ydata, yout** and **trans**
- **newpk** indicates whether or not **shape** is a new peakshape
- **iter** is the iteration count

Output arrays:
- **yout**, filled between **0** and **n2**

Output variables:
- **n** = amount of array space used
  - NOTE: if **n** is returned negative, abs(n) = amount of array space needed (but not available). Operation not successful.
- **newpk** = 0 if **trans** was successfully loaded from **shape**
- **nfwhm** = full-width-at-half-maximum of peakshape
- **sigma** = RMS noise in **ydata**

Function return values:
- **rzrpsm** = 0 if successful
  - If **rzrpsm** < 0, error occurred
  - Use **rzrerr** (page 174) to obtain error text

Description of variables

ydata on *input* is the *raw data array.* It should contain the raw data between data points **0** and **n2**. **ydata** *will* be altered outside this range.

ydata must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

n2 is the *last location* of data in the **ydata** array. **n2** is to be furnished as *input.*

shape is an *input* array which holds the *peakshape of the narrowest spectral feature* in **ydata** which is of interest to the user. The relevant peakshape is located between data points **0** and **nl2** in **shape**.

nl2 is *input* and the *index of the last data point of the peakshape* in **shape**. We recommend that **nl2**+1 be at least 6⋆**nfwhm**, and that the peak be approximately centered in the (**0**,**nl2**) interval.

yout is the *output smoothed data array*. It will be smoothed between data points **0** and **n2**, and should be ignored outside this range.

yout must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

w is a *work array of size* **n** which will be used for computations. W must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

trans is an *array of size n* which will be used to house the Fourier transform of the peakshape. The amount of space used in **trans** is calculated in **rzprep**. See the discussion below for **n**.

trans is either empty or filled, depending on the parameter **newpk**. Whenever **newpk** = 1, it is assumed that the contents of **shape** have been altered, and **trans** is properly loaded by **rzrpsm**. When **newpk** = 0, it is expected that **trans** has not been changed since the last time it was filled. See the discussion below for **newpk**.

n is *input* as the *amount of space furnished* in the **yout**, **w**, and **trans** arrays.

The function **rzsizn** will calculate **n**, the minimum amount of space needed. The required size of **n** is determined by **n2** and by the width of the peak in the **shape** array. Obtain the minimum required **n** with this call:

n = rzsizn(n2,shape,nl2)

On *output*, **n** is the amount of *space used for the Fourier transforms* in the **yout**, **w**, and **trans** arrays. If **n** is negative on output, the amount of space furnished was inadequate, and no processing has taken place. If **n** is returned negative, then abs(**n**) is the amount of space needed in the above arrays.

The space required for the Fourier transform is always calculated in **rzprep**, described in Chapter 11. When **newpk** = 1, **rzprep** calculates the required size of the Fourier transform as the smallest power of two larger than (**n2**+1+3***nfwhm**). You may wish to calculate **n** in an alternate fashion. See Chapter 11.

NOTE: When **rzrpsm** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you with to process additional data with the same peakshape, you need not change either **newpk** or **n**, provided that (a) your peakshapes do not change, and (b) your input **ydata** sizes (**n2**+1) do not increase.

newpk on *input* is an *integer flag* set which should be initially set to 1. It informs the peakshape processor that a new peakshape is present in **shape**. The processor measures certain parameters of the new peakshape, and then fills the **trans** array with the Fourier transform of a properly shifted and scaled peakshape. When the peakshape processor finishes successfully, it will *output* **newpk** = **n**, where **n** is the actual space used in **trans**.

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1*. Whenever **rzrpsm** is called with **newpk** > 1, ensure that:

(a) The user wants to use the previous peakshape for the current processing, and **trans** is not changed.

(b) The size of the array needed to transform the new data set is no larger than the **n** used previously. If this second criterium is violated, the *output* value of **rzrpsm** will be **rzrpsm** = -2.

iter is an *input index* for the iteration loop. Set iter = 0 for the initial call only. The function distinguishes between iter = 0 and iter > 0. It will update **iter** automatically.

nfwhm is *output* as the number of *data points between the half-maxima* of the peakshape feature in **shape**.

sigma is *output* as the *standard deviation (root- mean-square) of the noise* which has been removed by the smoothing process.

2.6 Example using rzrpsm

The data file SPEC4 represents radio chromatography, where one is counting nuclear disintegrations in a flow cell, and the background counts are 50% of the total signal. The peakshape is derived from a strong peak in the same cell. The original data had more noise than you will see in the peakshape data file PEAK4. One really should average a lot of strong peaks to get a smooth representation. We only had one strong peak, so we smoothed it, and used it. Because the peakshapes in a flow cell are so asymmetric, it is important to use real data.

Data file: SPEC4
Peakshape file: PEAK4

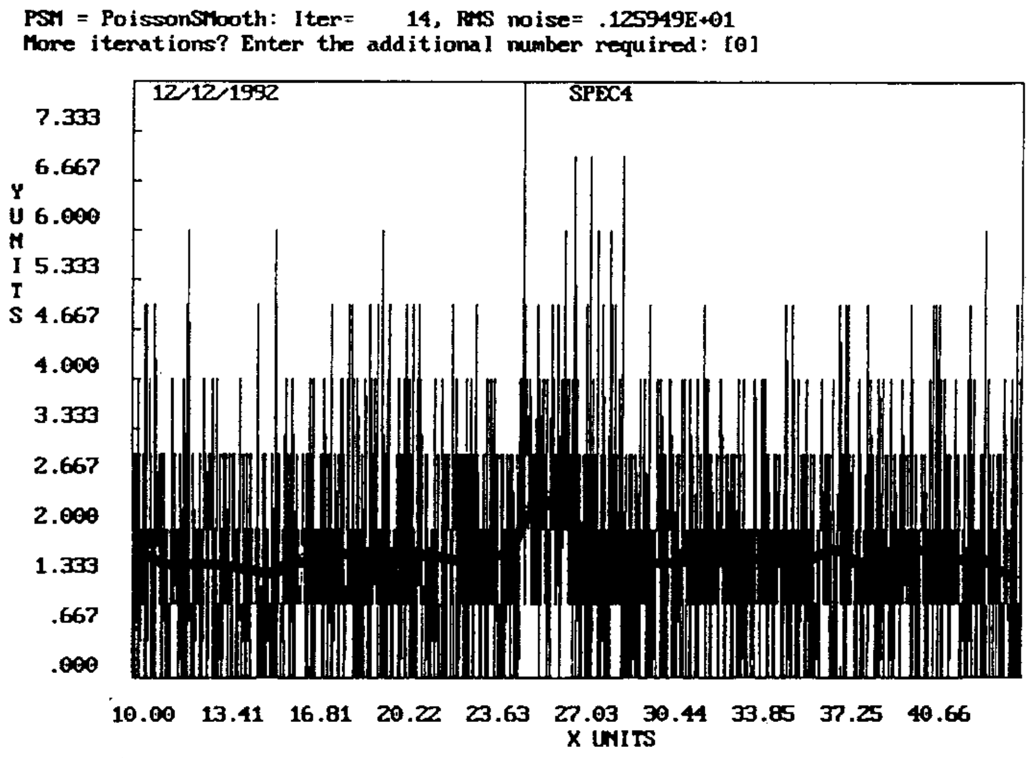
Using HANDLE:

```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
ESM=EntropySMooth. Smooths Normal (thermal/gaussian) noise. ME
PSM=PoissonSMooth. Smooths Poisson (counting) noise. ML.
NSM=NormalSMooth. Smooths Normal noise. ML.
DIV=RazorDivide. Calculates transmission spectra. ML.
ASH=RazorASharp. Enhances resolution. ML.
DEC=RazorDeconvolve. Maximum Entropy deconvolution. ME/Bayesian.
LUC=RazorLucy. Classic ML deconvolution.  ML.
DIF=RazorDerivative. Derivatives 0th-nth. Bayesian.
PIC=RazorPick. Finds peak positions for FIT. ML/Bayesian.
FIT=RazorFit. Fits model peaks to data.  ML.
BAS=RazorBase. Finds baseline. ME/Bayesian.
QBA=RazorQuickBase. Finds baseline.
EDG=RazorEdge. Fits baseline to lower edge of data.
NOI=RazorNoise. Finds noise spectrum. ML.
GEN=Generates synthetic peakshape.
SAV=Save result, QUI=Quit.
Choose an operation (3 uppercase characters): PSM

Enter name of spectrum: SPEC4
Enter name of peakshape: PEAK4

Entering RZRPSM with iter=0.  Please wait for setup...
At iter 1 the RMS noise is 1.880
     ........
     ........
At iter 15 the RMS noise is 1.258
More iterations? Enter the additional number required [0]:
```

```
The standard deviation of the removed noise is 1.257
The FWHM of the peakshape is 141
The size of the array space used was 4096
RESULT MAY BE SAVED TO A FILE
```



```
PSM = PoissonSMooth: Iter=     14, RMS noise= .125949E+01
More iterations? Enter the additional number required: [0]
```

2.7 rzrnsm — Razor Normal Smooth

Razor Normal Smooth (**rzrnsm**) provides a Maximum Likelihood estimate of a noise-free parent spectrum. The observed spectrum is a single a noisy example drawn from this parent spectrum. The noise is assumed to come from a Normal distribution.

The required user input for **rzrnsm** is:

- Data array. *The input data set must be positive,* as is appropriate for data with Poisson noise. **It is the user's responsibility to remove the correct offset from the raw data before using rzrnsm.**
- Peakshapes - either true or estimated. It is not critical that the user choose an exact peakshape for **rzrnsm**. When all the peaks in the data are not the same, the user should select a smooth peakshape characteristic of the *narrowest* feature of interest in the data.

Processing notes:

- The function produces an estimated parent spectrum which is constrained to be positive.
- This function is iterative, and therefore takes considerably more time than **rzresm**.

Programmer notes:

- **rzrnsm** requires 4 full-sized arrays. If space is a problem, see Section 11.2.
- Set iter = 0 for the initial call. **rzrnsm** will then maintain iter for you. **rzrnsm** needs 15 to 25 iterations. Some peakshapes converge faster. When the peakshape is Gaussian, the convergence is faster than when the peakshape is Lorentzian.

long rzrnsm(float ydata[ ], long n2, float shape[ ], long nl2,
float yout[ ], float w[], float z[], float trans[ ], long *n, long *newpk,
long *iter, long *nfwhm, double *sigma)

Input arrays which must be filled:
- **ydata**, filled between **0** and **n2**, length **n2 + 1**
- **shape**, filled between **0** and **nl2**
  - NOTE: If **newpk** = 0, **shape** will not be used.
  - NOTE: **shape** will be read only, not altered.

Additional arrays to be furnished:
- **yout**, length **n**
- **w**, length **n**
- **z**, length **n**
- **trans**, length **n**

Input variables: **n2, nl2, n, newpk**
- **n2** is the index of the last data values in **ydata**
- **nl2** is the index of the last data value in **shape**
- **n** is the size of arrays **ydata, yout** and **trans**
- **newpk** indicates whether or not **shape** is a new peakshape
- **iter** is the iteration count

Output arrays:
- **yout**, filled between **0** and **n2**

Output variables:
- **n** = amount of array space used
  - NOTE: if **n** is returned negative, abs(n) = amount of array space needed (but not available). Operation not successful.
- **newpk** = 0 if **trans** was successfully loaded from **shape**
- **nfwhm** = full-width-at-half-maximum of peakshape
- **sigma** = RMS noise in **ydata**

Function return values:
- **rzrnsm** = 0 if successful
  - If **rzrnsm** < 0, error occurred
  - Use **rzrerr** (page 174) to obtain error text

Description of variables

ydata on *input* is the *raw data array*. It should contain the raw data between data points **0** and **n2**. **ydata will not** be altered outside this range.

ydata must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

n2 is the *last location* of data in the **ydata** array. **n2** is to be furnished as *input*.

shape is an *input* array which holds the *peakshape of the narrowest spectral feature* in **ydata** which is of interest to the user. The relevant peakshape is located between data points **0** and **nl2** in **shape**.

nl2 is *input* and the *index of the last data point of the peakshape* in **shape**. We recommend that **nl2**+1 be at least 6⋆**nfwhm**, and that the peak be approximately centered in the (**0**,**nl2**) interval.

yout is the *output smoothed data array.* It will be smoothed between data points **0** and **n2**, and should be ignored outside this range.

yout must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

w is a *work array of size* **n** which will be used for computations. W must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

z is a *work array of size* **n** which will be used for computations. W must have a minimum size equal to the smallest power of two larger than (**n2**+1+3***nfwhm**). See the discussion below for **n**.

trans is an *array of size n* which will be used to house the Fourier transform of the peakshape. The amount of space used in **trans** is calculated in **rzprep**. See the discussion below for **n**.

trans is either empty or filled, depending on the parameter **newpk**. Whenever **newpk** = 1, it is assumed that the contents of **shape** have been altered, and **trans** is properly loaded by **rzrnsm**. When **newpk** = 0, it is expected that **trans** has not been changed since the last time it was filled. See the discussion below for **newpk**.

n is *input* as the *amount of space furnished* in the **yout**, **w**, and **trans** arrays.

The function **rzsizn** will calculate **n**, the minimum amount of space needed. The required size of **n** is determined by **n2** and by the width of the peak in the **shape** array. Obtain the minimum required **n** with this call:

n = rzsizn(n2,shape,nl2)

On *output*, **n** is the amount of *space used for the Fourier transforms* in the **yout**, **w**, and **trans** arrays. If **n** is negative on output, the amount of space furnished was inadequate, and no processing has taken place. If **n** is returned negative, then abs(**n**) is the amount of space needed in the above arrays.

The space required for the Fourier transform is always calculated in **rzprep**, described in Chapter 11. When **newpk** = 1, **rzprep** calculates the required size of the Fourier transform as the smallest power of two larger than (**n2**+1+3***nfwhm**). You may wish to calculate **n** in an alternate fashion. See Chapter 11.

NOTE: When **rzrnsm** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you with to process additional data with the same peak-shape, you need not change either **newpk** or **n**, provided that (a) your peakshapes do not change, and (b) your input **ydata** sizes (**n2**+1) do not increase.

newpk on *input* is an *integer flag* set which should be initially set to 1. It informs the peakshape processor that a new peakshape is present in **shape**. The processor measures certain parameters of the new peakshape, and then fills the **trans** array with the Fourier transform of a properly shifted and scaled peakshape. When the peakshape processor finishes successfully, it will *output* **newpk** = **n**, where **n** is the actual space used in **trans**.

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1.* Whenever **rzrnsm** is called with **newpk** > 1, ensure that:

(a) The user wants to use the previous peakshape for the current processing, and **trans** is not changed.

(b) The size of the array needed to transform the new data set is no larger than the **n** used previously. If this second criterium is violated, the *output* value of **rzrnsm** will be **rzrnsm** = -2.

iter is an *input index* for the iteration loop. Set iter = 0 for the initial call only. The function distinguishes between iter = 0 and iter > 0. It will update **iter** automatically.

nfwhm is *output* as the number of *data points between the half-maxima* of the peakshape feature in **shape**.

sigma is *output* as the *standard deviation (root- mean-square) of the noise* which has been removed by the smoothing process.

2.8 Example using rzrnsm

Data file: SPEC2

Peakshape file: PEAK2

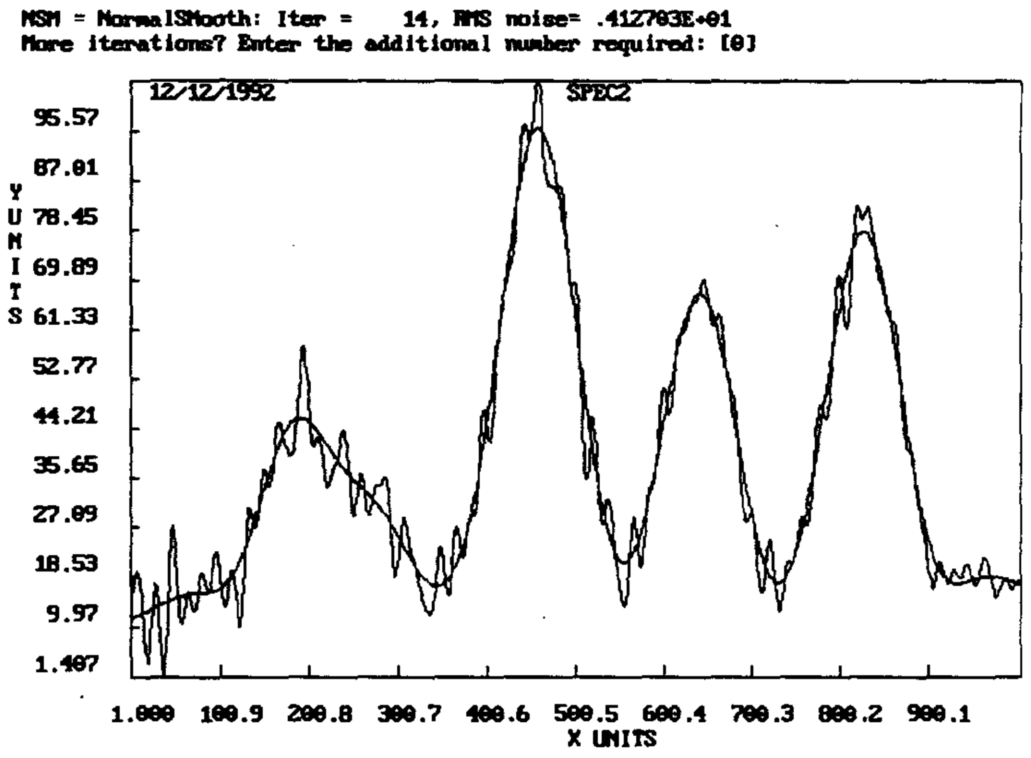
Using HANDLE:

```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
ESM=EntropySMooth. Smooths Normal (thermal/gaussian) noise. ME
PSM=PoissonSMooth. Smooths Poisson (counting) noise. ML.
NSM=NormalSMooth. Smooths Normal noise. ML.
DIV=RazorDivide. Calculates transmission spectra. ML.
ASH=RazorASharp. Enhances resolution. ML.
DEC=RazorDeconvolve. Maximum Entropy deconvolution. ME/Bayesian.
LUC=RazorLucy. Classic ML deconvolution.  ML.
DIF=RazorDerivative. Derivatives 0th-nth. Bayesian.
PIC=RazorPick. Finds peak positions for FIT. ML/Bayesian.
FIT=RazorFit. Fits model peaks to data.  ML.
BAS=RazorBase. Finds baseline. ME/Bayesian.
QBA=RazorQuickBase. Finds baseline.
EDG=RazorEdge. Fits baseline to lower edge of data.
NOI=RazorNoise. Finds noise spectrum. ML.
GEN=Generates synthetic peakshape.
SAV=Save result, QUI=Quit.
Choose an operation (3 uppercase characters): NSM

Enter name of spectrum: SPEC2
Enter name of peakshape: PEAK2

Entering RZRNSM with iter=0.  Please wait for setup...
At iter 1 the RMS noise is 5.5789
     ........
     ........
At iter 15 the RMS noise is 4.190
More iterations? Enter the additional number required [0]:

The standard deviation of the removed noise is 4.190
The FWHM of the peakshape is 80
The size of the array space used was 2048
RESULT MAY BE SAVED TO A FILE
```



```
NSM = NormalSMooth: Iter =     14, RMS noise= .412703E+01
More iterations? Enter the additional number required: [0]
```

2.9 Maximum Likelihood Smoothing — Theory

Maximum Likelihood techniques are used by chemists and spectroscopists every day. Methods such as least-square peak fitting, linear regression, and even the simple formula for averaging a set of scans, all can be derived from Maximum Likelihood principles. We have used Maximum Likelihood methods to derive a new, statistically sound method for smoothing.

The remaining sections of this chapter are organized so that our readers may understand the concepts, while skipping the mathematical sections, if they choose. We discuss our premise in Section 2.10, and the Maximum Likelihood principle in Section 2.11. The mathematical parts and equations in Section 2.12, where we set up the basic equations, and Section 2.13, where we show which forms of the basic equations are being solved by the **RZRESM, RZRPSM and RZRNSM** algorithms.

2.10 The purpose of a smoothing formula

Smoothing prescriptions should answer the question: what would the data look like if the observer could average many, many scans? If one could make many measurements of a sample, and stack scans, the underlying features of the physical process would be revealed, without any sacrifice in resolution. The final averaged smooth curve, called the **parent spectrum**, is the desired result.

The purpose of a good smoothing formula should be to provide an estimate of the parent spectrum from which a particular noisy sample (spectrum, chromatogram) was drawn. This estimate should be formulated from physical knowledge about the experiment which can be agreed upon in advance. The formula should not contain any arbitrarily chosen parameters.

The idea behind Maximum Likelihood smoothing is simple: Each scan (of a spectrum), and each run (in chromatography), can be thought of as a single noisy sample drawn from some parent spectrum. Maximum Likelihood estimates the parent spectrum by answering the question: "What is the most probable spectrum, or chromatogram, buried under all this noise?"

Maximum Likelihood smoothing derives its power from the a priori information known to the observer. When an observer looks at noisy data, he usually has a fairly good idea of what is 'real', and what is noise. His judgement is governed by his intuitive knowledge of what a 'real peak' looks like. In fact, it is precisely this intuitive knowledge of peakshapes which allows him to select a parameter such as a filter width.

In Maximum Likelihood smoothing, we take the a priori knowledge of the peakshape, as well as a priori information about the type of noise seen in the data, and cast both into a mathematical framework. The result is a smoothing formula which is *optimum*, in the sense that it provides the *best possible* estimate of what we would see if we could average our data for a much longer time.

2.11 Maximum Likelihood Foundation

Suppose we have measured a data set $\{y_1, y_2, ...y_n\}$. The individual values in this data set, the y_i, may be absorbances at different frequencies, or they may be radioactive disintegrations counted as a function of time, or whatever is being measured. We really want to know the values of the data set $\{z_1, z_2, ...z_n\}$, where each z_i is a mean of many measurements of y_i. In other words, the set $\{z_1, z_2, ...z_n\}$ is the high signal/noise result we would obtain if we could average for a long time. The relation between the set $\{y_1, y_2, ...y_n\}$ and the set $\{z_1, z_2, ...z_n\}$ is $y_i = z_i + n_i$, where n_i are the noise fluctuations. We will use Maximum Likelihood to estimate $\{z_1, z_2, ...z_n\}$.

What does it mean to say we will estimate $\{z_1, z_2, ...z_n\}$? We have only one data sample $\{y_1, y_2, ...y_n\}$, and so we cannot estimate $\{z_1, z_2, ...z_n\}$ by averaging. Furthermore, we do not presume that $\{z_1, z_2, ...z_n\}$ can be modeled by some analytic function (i.e., a polynomial, or a sum of 14 gaussians, etc.) whose parameters we might obtain through a least-squares peak-fitting technique. Instead, we obtain our estimate as follows: We require that the estimate conform to certain a priori knowledge about the noise characteristics, and about the instrument. Beyond that, we assume that our observations have occurred in a very ordinary room, in an ordinary corner of the universe.

Here is the kernal of Maximum Likelihood: We assume that the particular data sample we have observed, the set $\{y_1, y_2, ...y_n\}$, is a typical, representative sample. This data sample is *so ordinary* that there is no statistical difference between this one and many thousands of other noisy data sets which might have occurred. The sample is therefore representative of the *most likely* statistical behavior. Consequently, we will estimate the parent spectrum $\{z_1, z_2, ...z_n\}$ by writing an equation which describes the probability for the data set $\{y_1, y_2, ...y_n\}$, and then we will maximize that probability, under conditions which also satisfy all known a priori constraints.

2.12 Smoothing Equations

We want to set up an equation which describes the probability of obtaining the data set $\{y_1, y_2, ...y_n\}$, in terms of the parent spectrum $\{z_1, z_2, ...z_n\}$. For a given parent spectrum, the probability p for the sample $\{y_1, y_2, ...y_n\}$ is determined by the probability distributions for the noise $\{n_1, n_2, ...n_n\}$. p is usually called the likelihood function.

We now incorporate our a priori knowledge about the nature of the noise. Usually we know the statistical characteristics of the noise in an experiment. When we are counting individual particles, such as beta or gamma particles from radioactive decay, x-rays, or photons in a low-light situation, the fluctuations n_i in the data usually come from Poisson distributions. Poisson noise has the property that the root-mean-square (rms) noise amplitude is proportional to the square root of the signal amplitude. On the other hand, when detector noise dominates, as in system noise in amplifiers, in thermal detectors, and in many other cases, then the noise is independent of the signal amplitude, additive, and describable by a Normal distribution. There are cases where neither Normal nor Poisson statistics apply, as in photomultiplier dark current. Whatever the noise, we must write an equation which incorporates its statistics.

We are ready to write an equation for the probability of obtaining our observed data $\{y_1, y_2, ...y_n\}$. From a parent spectrum $\{z_1, z_2, ...z_n\}$, we have drawn a data set $\{y_1, y_2, ...y_n\}$ containing data points $y_1, y_2, ...y_n$, where $y_i = z_i + n_i$. The n_i are the noise values.

If the noise n_i is random, and additive, with a Normal distribution, then the probability for y_i is

$$p(y_i \mid z_i) = \frac{1}{\sqrt{(2\pi)}\sigma_i}\exp[-\frac{(y_i - z_i)^2}{2\sigma_i^2}].$$

If the noise n_i is random noise with a Poisson distribution, then the probability for y_i is

$$p(y_i \mid z_i) = \frac{{z_i}^{y_i} e^{-z_i}}{y_i!}.$$

Assume that the noise n_i is uncorrelated with the noise n_j, for all i, j. Then the likelihood of observing the set $\{y_1, y_2, ...y_n\}$ is the product of the probabilities for each of the y_i:

$$p(y_1, ...y_n \mid z_1, ..., z_n) = \prod_{i=1}^{n} p(y_i \mid z_i)$$

For Normal noise, this becomes

$$p(y_1, ...y_n \mid z_1, ..., z_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_i}\exp[-\frac{(y_i - z_i)^2}{2\sigma_i^2}].$$

For Poisson noise,

$$p(y_1, ...y_n \mid z_1, ..., z_n) = \prod_{i=1}^{n} \frac{{z_i}^{y_i} e^{-z_i}}{y_i!}.$$

The Maximum Likelihood prescription says we must maximize p. The maximization is be done under a set of constraints. An important constraint is our knowledge of the peakshapes. We assume that the parent spectrum is composed of many individual peaks, of known shapes. However, we make absolutely no assumption about how many peaks there are, how large they are, or where they are. In fact, we allow as many peaks as there are data points in the observed spectrum or chromatogram! We may also have additional knowledge about the parent spectrum, e.g., often the parent spectrum cannot be negative. Any such constraints are allowed. In fact, the more we known about the underlying smooth spectrum, or about the instrument, or the measurement bias, or the noise, etc, the better will be our estimate of the parent spectrum.

We now proceed in one of two ways:
(1) We maximize the probability

$$\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_\mathrm{n} \mid \mathrm{z}_1, ..., \mathrm{z}_\mathrm{n}),$$

by looking for the set $\{\mathrm{z}\}$ which maximizes p, and also satisfies the conditions $\mathrm{z} = \mathrm{o} \otimes \mathrm{s}$, where s is the characteristic shape of all single peaks, o is the object function, and $\otimes$ denotes convolution. In certain cases, such as photon counting, o must be positive.
(2) We maximize the probability

$$\mathrm{p}(\mathrm{z}_1, ..., \mathrm{z}_\mathrm{n} \mid \mathrm{y}_1, ..., \mathrm{y}_\mathrm{n}),$$

by invoking Bayes Rule. This is now commonly called the Bayesian method. It used to be called the MAP (Maximum A Posteriori) method.

Bayes Rule says that the probability $\mathrm{p}(\mathrm{z}_1, ..., \mathrm{z}_\mathrm{n} \mid \mathrm{y}_1, ..., \mathrm{y}_\mathrm{n})$ is related to the probability $\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_\mathrm{n} \mid \mathrm{z}_1, ..., \mathrm{z}_\mathrm{n})$ through

$$\mathrm{p}(\mathrm{z}_1, ..., \mathrm{z}_\mathrm{n} \mid \mathrm{y}_1, ..., \mathrm{y}_\mathrm{n}) = \frac{\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_\mathrm{n} \mid \mathrm{z}_1, ..., \mathrm{z}_\mathrm{n})\mathrm{p}(\mathrm{z}_1, ...\mathrm{z}_\mathrm{n})}{\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_\mathrm{n})}.$$

In order to solve the equation, we must also provide the a priori probabilities $\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_\mathrm{n})$ for our observed spectrum and $\mathrm{p}(\mathrm{z}_1, ..., \mathrm{z}_\mathrm{n})$ for all parent spectrums. Clearly, the probability for our single observation is $\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_\mathrm{n}) = 1$.

A common expression of the a priori probability for the parent spectrum is given by the combinatorial probability

$$\mathrm{p}(\mathrm{z}_1, ..., \mathrm{z}_\mathrm{n}) = \frac{(\mathrm{z}_1 + \mathrm{z}_2 + ... + \mathrm{z}_\mathrm{n} = \mathrm{N})!}{\mathrm{z}_1!\mathrm{z}_2!...\mathrm{z}_\mathrm{n}!}$$

.

The combinatorial probability states that if you have N items which are to be sorted into n boxes, the probability of obtaining an arrangement where z_1 are in the first box, z_2 are in the second box, etc, is proportional to the number of combinations of the N distinct items which give box occupation numbers $\{z_1, z_2, ...z_n\}$.

When all possible parent spectrums have the same a priori probability,

$$\mathrm{p}(\mathrm{z}_1, ..., \mathrm{z}_\mathrm{n}) = \mathrm{constant}$$

, then the solution will be the same as that for case (1) above.

Again, we will require that the condition $\mathrm{z} = \mathrm{o} \otimes \mathrm{s}$, where s is the peakshape and o is the object function, is satisfied. In some cases, we may also require that o is positive.

2.13 Three solutions

Razor Library contains three separate smoothing methods, corresponding to three different solutions of the equations posed in Section 2.12. The three methods, and the equations which they solve, are given here.

Razor Entropy Smooth — RZRESM is both a Maximum Entropy and a Bayesian method. It maximizes the probability

$$p(z \mid y) = \{\prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_i} \exp[-\frac{(y_i - z_i)^2}{2\sigma_i^2}]\}\{\frac{(z_1 + \ldots + z_n = N)!}{z_1!z_2!\ldots z_n!}\},$$

under the constraint that $z = o \otimes s$. The Maximum Entropy character becomes evident if we take the logarithm of p,

$$\ln(p(z \mid y)) = \sum_{i=1}^{n}(-\frac{(y_i - z_i)^2}{2\sigma_i^2} - z_i \ln z_i) + \text{constant terms}$$

.

The term $-\sum z_i ln z_i$ is the same expression as the Shannon entropy for the spectrum $\{z_1, z_2, \ldots z_n\}$. Note that maximizing $ln(p(z \mid y))$ is the same as maximizing $p(z \mid y)$, because the probability $p(z \mid y)$ is always positive.

Razor Poisson Smooth — RZRPSM is an iterative solution to the Poisson probability distribution equation

$$p(y \mid z) = \prod_{i=1}^{n} \frac{{z_i}^{y_i} e^{-z_i}}{y_i!} = \text{maximum},$$

under the constraints that $z = o \otimes s$, and that **o** is positive.

Razor Normal Smooth — RZRNSM is an iterative solution to the Normal probability distribution equation

$$p(y \mid z) = \prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_i} \exp[-\frac{(y_i - z_i)^2}{2\sigma_i^2}] = \text{maximum},$$

under the constraints that $z = o \otimes s$, and that **o** is positive.

2.14 Limitations of rzrpsm and rzrnsm

rzrpsm and **rzrnsm** are iterative algorithms which search for the maximum of the probability expressions shown in Section 2.13. They are not particularly fast, but we make no apology for that here. The specific algorithms were chosen for their stability, and for immunity to such details as cumulative truncation error.

For **rzrpsm**, 15 - 25 iterations are adequate for most data we have encountered. However, if you find that **rzrpsm** does not provide you with a satisfactory result, we request that you contact us. We would appreciate your sending us your difficult data.

rzrnsm is yet another story. Because it is very slow to converge, we think you might decide to use **rzresm** for all your Normal data, just as we do.