

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

©Copyright 1991 - 1998 by Spectrum Square Associates, Inc., Ithaca, NY 14850
All rights reserved.

COPYRIGHT: This software is protected by both United States copyright law and international treaty provisions. No part of this publication may be reproduced or transmitted in any form or by any means, without prior written consent of Spectrum Square Associates.

SPECTRUM SQUARE LICENSE AGREEMENT: This is a legal agreement between the user of the enclosed software and Spectrum Square Associates, Inc. BY OPENING THE SEALED DISK PACKAGE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS, PLEASE RETURN THE UNOPENED DISK PACKAGE AND THE ACCOMPANYING MANUAL, FOR A FULL REFUND.

GRANT OF LICENSE: Spectrum Square Associates grants you the right to use the enclosed software on a single computer. You may make copies of the software for backup purposes, provided that you label all copies with the copyright notice.

You may not distribute any software incorporating any portions of Razor Library source code, object modules, or library files, without obtaining a separate License Agreement from Spectrum Square Associates. Royalties will apply.

DISCLAIMER OF WARRANTY: THIS SOFTWARE AND MANUAL ARE SOLD "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OR MERCHANTABILITY. The seller's salespersons may have made statements about this software. any such statements do not constitute warranties and shall not be relied on by the buyer in deciding whether to purchase this software.

THIS SOFTWARE IS SOLD WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES WHATSOEVER. Because of the diversity of conditions under which the software may be used, no warranties of fitness for a particular purpose is offered. THE USER IS ADVISED TO TEST THE SOFTWARE THOROUGHLY BEFORE RELYING ON IT. THE USER MUST ASSUME THE ENTIRE RISK OF USING THE SOFTWARE. Any liability of seller or manufacturer will be limited exclusively to product replacement or refund of the purchase price.

IN NO EVENT SHALL SPECTRUM SQUARE ASSOCIATES OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PRODUCT, EVEN IF SPECTRUM SQUARE ASSOCIATES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

GOVERNING LAW: This License Agreement shall be governed by the laws of the State of New York.

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:
 - `rzresm` — `RazorEntropySmooth` — page 8
 - `rzrpsm` — `RazorPoissonSmooth` — page 14
 - `rzrnsm` — `RazorNormalSmooth` — page 20
 - `rzrdiv` — `RazorDivide` — page 33
 - `rzrash` — `RazorASharp` — page 46
 - `rzrdec` — `RazorDeconvolve` — page 52
 - `rzrluc` — `RazorLucy` — page 59
 - `rzrstr` — `RazorStrip` — page 74
 - `rzrdif` — `RazorDerivative` — page 79
 - `rzrpic` — `RazorPick` — page 89
 - `rzrfit` — `RazorFit` — page 97
 - `rzrbas` — `RazorBase` — page 139
 - `rzrqba` — `RazorQuickBase` — page 147
 - `rzredg` — `RazorEdge` — page 152
 - `rzrcut` — `RazorCut` — page 155
 - `rzrnoi` — `RazorNoise` — page 158
- 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 **rzrpc** 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 **rzrpc** and **rzrbas**. See Page 177, and **handle.for**.
- **rzrpc** and **rzrbas** will automatically search for *negative peaks*, if a negative peak is presented in the **shape** array.

Contents

1	Razor Library Description	1
1.1	Advanced Statistical Functions	1
1.2	Principal Razor functions	2
1.3	User input and programmer control	3
1.4	Example source code	3
1.5	Source for service functions	4
2	RazorSmooth — rzresm/rzrpsm/rzrnsm	5
2.1	Smoothing which Preserves Resolution	5
2.2	Which one to use?	5
2.3	rzresm — Razor Entropy Smooth	7
2.4	Example using rzresm	11
2.5	rzrpsm — Razor Poisson Smooth	13
2.6	Example using rzrpsm	17
2.7	rzrnsm — Razor Normal Smooth	19
2.8	Example using rzrnsm	23
2.9	Maximum Likelihood Smoothing — Theory	25
2.10	The purpose of a smoothing formula	25
2.11	Maximum Likelihood Foundation	26
2.12	Smoothing Equations	26
2.13	Three solutions	29
2.14	Limitations of rzrpsm and rzrnsm	30
3	RazorDivide — rzddiv	31
3.1	Noise Reduction for Ratio Spectra	31
3.2	rzddiv	31
3.3	Example using rzddiv	36
3.4	Reducing Noise in Transmission Spectra	39
4	RazorSharp — rzdash/rzddec/rzdluc	43
4.1	Resolution Enhancement without Artifacts	43
4.2	rzdash — RazorASHarp	45

4.3	Example using rzrash	49
4.4	rzrdec — RazorDEConvolve	51
4.5	Example using rzrdec	56
4.6	rzrluc — RazorLUCy	58
4.7	Example using rzrluc	62
4.8	Statistically Sound Restoration	64
4.9	The Bayesian Principle	64
4.10	How Bayesian/Maximum Likelihood/Maximum Entropy Restoration Works	64
4.11	Equations used by rzrash and rzrdec and rzrluc	66
4.11.1	Maximum Likelihood Restoration	66
4.11.2	Bayesian and Maximum Entropy Restoration	67
4.11.3	Razor Library's two restoration methods	67
4.11.4	rzrdec solution	69
4.11.5	rzrluc solution	69
4.11.6	rzrash solution	70
4.12	What about Fourier deconvolution?	71
4.13	A Final Word of Advice	72
4.14	rzrstr — RazorStrip	73
5	RazorDerivative — rzrdif	77
5.1	A Fundamental Approach to Derivatives	77
5.2	Example using rzrdif	82
5.3	Equations of Bayesian Derivatives	84
6	RazorPick — rzrpc	87
6.1	Accurate Peak-Picking for Merged Peaks	87
6.2	rzrpc	88
6.3	Example using rzrpc	93
7	RazorFit — rzrfit	95
7.1	Accurate Peak Areas, with Confidence Limits	95
7.2	rzrfit	96
7.3	First Example using rzrfit	105
7.4	Second Example using rzrpc and rzrfit	109
7.5	Third Example using rzrbas and rzrfit	114
7.6	The RazorFit algorithm	121
7.7	The RazorFit model	121
7.8	RazorFit and Maximum Likelihood	122
7.9	Downhill to a minimum	124
7.10	Confidence Limits	126
7.11	Limitations of RazorFit	126

8	Peakshape Catalog	127
8.1	Captured DataPeak	128
8.2	Gaussian	129
8.3	Lorentzian	129
8.4	Sum Gaussian + Lorentzian	129
8.5	Product Gaussian*Lorentzian	130
8.6	Asymmetric Gaussian	130
8.7	Asymmetric Lorentzian	131
8.8	Symmetric and Asymmetric Pearson ⁷	131
8.9	Log Normal	132
8.10	Baseline types	134
9	Baselines — rzrbas/rzrqba/rzredg/rzrcut	135
9.1	Baseline Fitting and Removal	135
9.1.1	RazorBase	135
9.1.2	RazorQuickBase and RazorEdge	136
9.2	rzrbas	137
9.3	Example using rzrbas	144
9.4	rzrqba	146
9.5	Example using rzrqba	149
9.6	rzredg	151
9.7	rzrcut	154
10	RazorNoise — rznri	157
10.1	rznri	157
10.2	Example using rznri	161
11	Service Functions	163
11.1	Fourier transforms — for speed	163
11.2	Transform padding — rzprep	164
11.3	rzparm	170
11.4	rzsizn tells array sizes.	172
11.5	Error messages from rzrerr	174
11.6	rzpkst - Sorts peaks from rzrpic/rzrbas	175
11.7	rzdfil - Loads peaks from rzrpic/rzrbas into datmat	177
11.8	rzrxpk - Removes baseline, smooths peakshape	179
11.9	New peakshapes	180

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 Calctm*, *Lab Calctm*, and *GRAMS/386tm*.

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 capabilities 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.

— **rzrpick**

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 (rznsm), 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 *nfw hm, 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, $\text{abs}(n)$ = amount of array space needed (but not available). Operation not successful.

newpk = **n** if **trans** was successfully loaded from **shape**

nfw hm = 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 $(n2+1+3*nfw hm)$. 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**.

n12 is *input* and the *index of the last data point of the peakshape* in **shape**. We recommend that **n12+1** be at least $6 \cdot \mathbf{nfwhm}$, and that the peak be approximately centered in the $(\mathbf{0}, \mathbf{n12})$ 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 $(\mathbf{n2} + 1 + 3 \cdot \mathbf{nfwhm})$. 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 **rsizn** 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 = rsizn(n2,shape,n12)
```

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 $\text{abs}(\mathbf{n})$ 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 $(\mathbf{n2} + 1 + 3 \cdot \mathbf{nfwhm})$. 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 wish 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 $(\mathbf{n2} + 1)$ 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 $\text{newpk} > 1$* . Whenever **rzresm** is called with $\text{newpk} > 1$, 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.

nfw 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
```

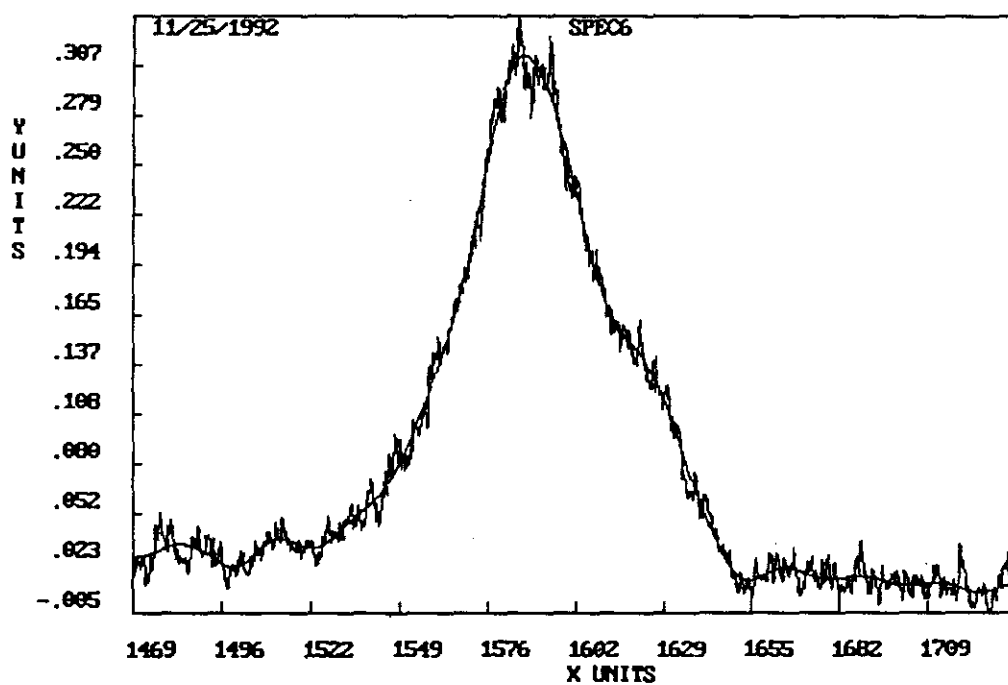
```
The FWHM of the peakshape is 149
```

```
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 rzspsm — Razor Poisson Smooth

Razor Poisson Smooth (**rzspsm**) 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 **rzspsm** 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 rzspsm.**
- Peakshapes - either true or estimated. It is not critical that the user choose an exact peakshape for **rzspsm**. 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 **rzsresm**.

Programmer notes:

- **rzspsm** requires 4 full-sized arrays. If space is a problem, see Section 11.2.
- Set `iter = 0` for the initial call. **rzspsm** will then maintain `iter` for you. **rzspsm** 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, $\text{abs}(n)$ = 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 $6 \times \mathbf{nfwhm}$, and that the peak be approximately centered in the $(0, \mathbf{nl2})$ 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 $(\mathbf{n2} + 1 + 3 \times \mathbf{nfwhm})$. 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 $(\mathbf{n2} + 1 + 3 \times \mathbf{nfwhm})$. 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 **rzsize** 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 = rzsize(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 $\text{abs}(\mathbf{n})$ 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 $(\mathbf{n2} + 1 + 3 \times \mathbf{nfwhm})$. 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 wish 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 $(\mathbf{n2} + 1)$ 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.

nfw 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

```

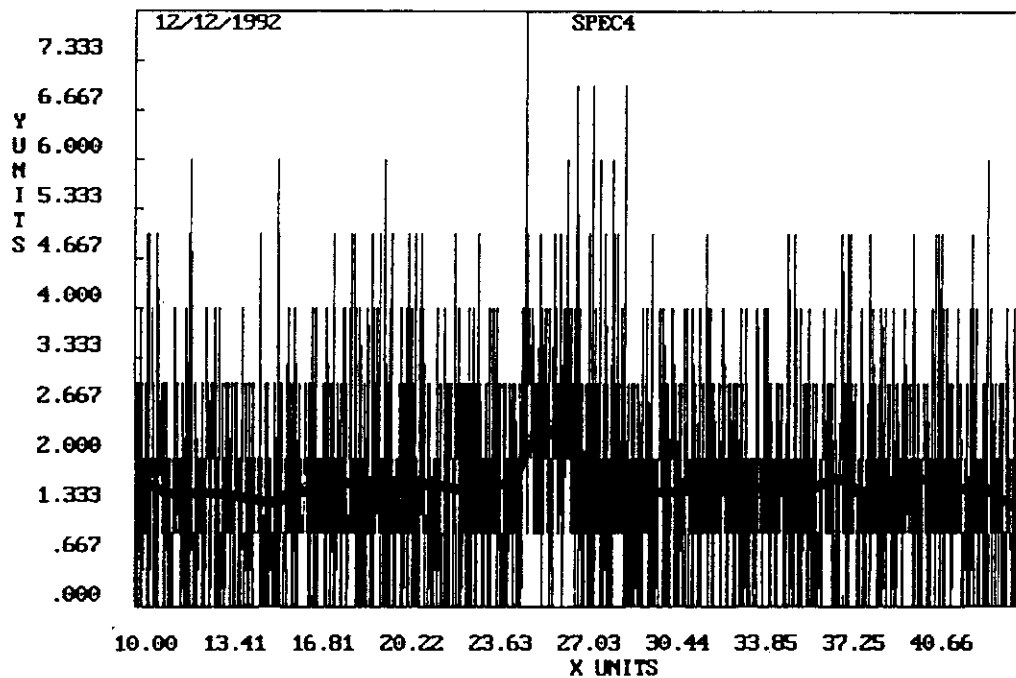
```

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 rznsm — Razor Normal Smooth

Razor Normal Smooth (**rznsm**) 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 **rznsm** 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 rznsm.**
- Peakshapes - either true or estimated. It is not critical that the user choose an exact peakshape for **rznsm**. 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 **rzesm**.

Programmer notes:

- **rznsm** requires 4 full-sized arrays. If space is a problem, see Section 11.2.
- Set `iter = 0` for the initial call. **rznsm** will then maintain `iter` for you. **rznsm** 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 rzrnsn( 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, $\text{abs}(n)$ = 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:

rzrnsn = 0 if successful

If **rzrnsn** < 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 $6 \times \mathbf{nfwhm}$, 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 $(\mathbf{n2}+1+3 \times \mathbf{nfwhm})$. 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 $(\mathbf{n2}+1+3 \times \mathbf{nfwhm})$. 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 $(\mathbf{n2}+1+3 \times \mathbf{nfwhm})$. 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 **rzrnsn**. 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 **rzsign** 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 = rzsign(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 $(\mathbf{n2}+1+3 \times \mathbf{nfwhm})$. You may wish to calculate **n** in an alternate fashion. See Chapter 11.

NOTE: When **rzrns**m returns after successful processing, it fills both **newpk** and **n** with the transform size. If you wish 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 **rzrns**m 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 **rzrns**m will be **rzrns**m = -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.

nfwh**m** 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 rznsm

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]:
```

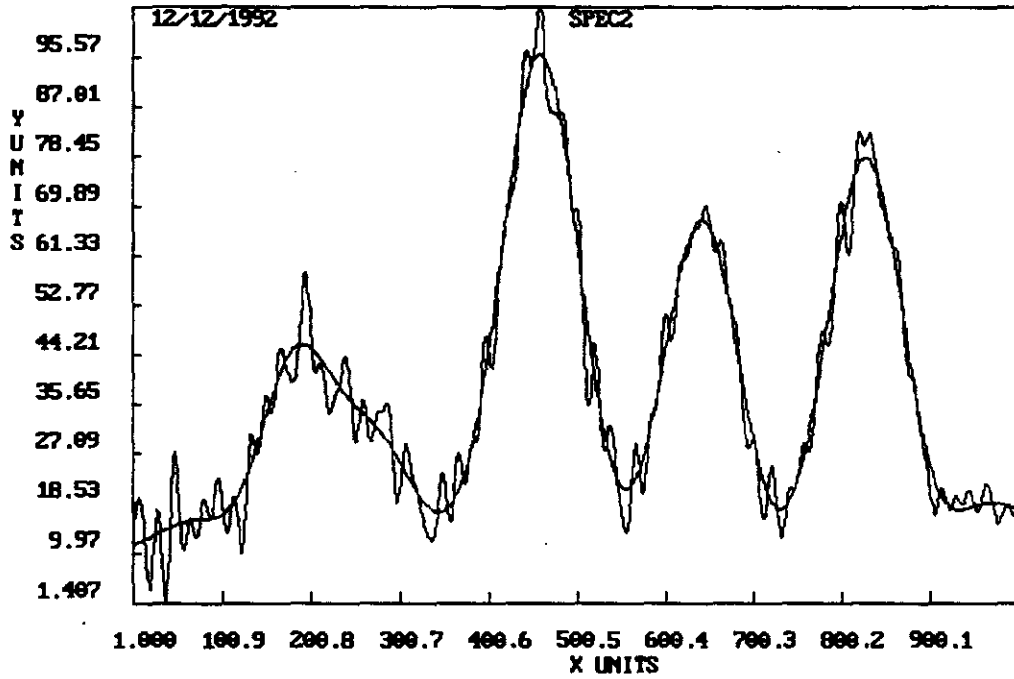
```
The standard deviation of the removed noise is 4.190
```

```
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= .412763E-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 $\{y_1, y_2, \dots, y_n\}$. 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 $\{z_1, z_2, \dots, z_n\}$, where each z_i is a mean of many measurements of y_i . In other words, the set $\{z_1, z_2, \dots, z_n\}$ is the high signal/noise result we would obtain if we could average for a long time. The relation between the set $\{y_1, y_2, \dots, y_n\}$ and the set $\{z_1, z_2, \dots, z_n\}$ is $y_i = z_i + n_i$, where n_i are the noise fluctuations. We will use Maximum Likelihood to estimate $\{z_1, z_2, \dots, z_n\}$.

What does it mean to say we will estimate $\{z_1, z_2, \dots, z_n\}$? We have only one data sample $\{y_1, y_2, \dots, y_n\}$, and so we cannot estimate $\{z_1, z_2, \dots, z_n\}$ by averaging. Furthermore, we do not presume that $\{z_1, z_2, \dots, z_n\}$ 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 kernel of Maximum Likelihood: We assume that the particular data sample we have observed, the set $\{y_1, y_2, \dots, y_n\}$, 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 $\{z_1, z_2, \dots, z_n\}$ by writing an equation which describes the probability for the data set $\{y_1, y_2, \dots, y_n\}$, 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 $\{y_1, y_2, \dots, y_n\}$, in terms of the parent spectrum $\{z_1, z_2, \dots, z_n\}$. For a given parent spectrum, the probability \mathbf{p} for the sample $\{y_1, y_2, \dots, y_n\}$ is determined by the probability distributions for the noise $\{n_1, n_2, \dots, n_n\}$. \mathbf{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 $\{y_1, y_2, \dots, y_n\}$. From a parent spectrum $\{z_1, z_2, \dots, z_n\}$, we have drawn a data set $\{y_1, y_2, \dots, y_n\}$ containing data points y_1, y_2, \dots, 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

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

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

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

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

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

For Normal noise, this becomes

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

For Poisson noise,

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

The Maximum Likelihood prescription says we must maximize p . The maximization is to 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 know 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

$$\mathbf{p}(y_1, \dots, y_n \mid z_1, \dots, z_n),$$

by looking for the set $\{z\}$ which maximizes \mathbf{p} , and also satisfies the conditions $z = o \otimes s$, 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

$$\mathbf{p}(z_1, \dots, z_n \mid y_1, \dots, y_n),$$

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 $\mathbf{p}(z_1, \dots, z_n \mid y_1, \dots, y_n)$ is related to the probability $\mathbf{p}(y_1, \dots, y_n \mid z_1, \dots, z_n)$ through

$$\mathbf{p}(z_1, \dots, z_n \mid y_1, \dots, y_n) = \frac{\mathbf{p}(y_1, \dots, y_n \mid z_1, \dots, z_n)\mathbf{p}(z_1, \dots, z_n)}{\mathbf{p}(y_1, \dots, y_n)}.$$

In order to solve the equation, we must also provide the a priori probabilities $\mathbf{p}(y_1, \dots, y_n)$ for our observed spectrum and $\mathbf{p}(z_1, \dots, z_n)$ for all parent spectrums. Clearly, the probability for our single observation is $\mathbf{p}(y_1, \dots, y_n) = 1$.

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

$$\mathbf{p}(z_1, \dots, z_n) = \frac{(z_1 + z_2 + \dots + z_n = N)!}{z_1!z_2!\dots z_n!}$$

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 $\{z_1, z_2, \dots, z_n\}$.

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

$$\mathbf{p}(z_1, \dots, z_n) = \text{constant}$$

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

Again, we will require that the condition $z = o \otimes s$, 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

$$p(\mathbf{z} | \mathbf{y}) = \left\{ \prod_{i=1}^n \frac{1}{\sqrt{(2\pi)\sigma_i}} \exp\left[-\frac{(\mathbf{y}_i - \mathbf{z}_i)^2}{2\sigma_i^2}\right] \right\} \left\{ \frac{(\mathbf{z}_1 + \dots + \mathbf{z}_n = \mathbf{N})!}{\mathbf{z}_1! \mathbf{z}_2! \dots \mathbf{z}_n!} \right\},$$

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

$$\ln(p(\mathbf{z} | \mathbf{y})) = \sum_{i=1}^n \left(-\frac{(\mathbf{y}_i - \mathbf{z}_i)^2}{2\sigma_i^2} - \mathbf{z}_i \ln \mathbf{z}_i \right) + \text{constant terms}$$

The term $-\sum \mathbf{z}_i \ln \mathbf{z}_i$ is the same expression as the Shannon entropy for the spectrum $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$. Note that maximizing $\ln(p(\mathbf{z} | \mathbf{y}))$ is the same as maximizing $p(\mathbf{z} | \mathbf{y})$, because the probability $p(\mathbf{z} | \mathbf{y})$ is always positive.

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

$$p(\mathbf{y} | \mathbf{z}) = \prod_{i=1}^n \frac{\mathbf{z}_i^{\mathbf{y}_i} e^{-\mathbf{z}_i}}{\mathbf{y}_i!} = \text{maximum},$$

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

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

$$p(\mathbf{y} | \mathbf{z}) = \prod_{i=1}^n \frac{1}{\sqrt{(2\pi)\sigma_i}} \exp\left[-\frac{(\mathbf{y}_i - \mathbf{z}_i)^2}{2\sigma_i^2}\right] = \text{maximum},$$

under the constraints that $\mathbf{z} = \mathbf{o} \otimes \mathbf{s}$, and that \mathbf{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.

Chapter 3

RazorDivide — `rzrdiv`

3.1 Noise Reduction for Ratio Spectra

RazorDivide is designed for transmission spectra, which have two especially difficult noise problems. RazorDivide solves the problem of how to reduce the noise in these ratio spectra.

The first noise problem arises when one divides a noisy sample spectrum by its noisy reference, producing a transmission spectrum with more noise, and with different noise statistics. If both the sample and reference spectra contained additive random noise from Gaussian distributions, the resultant transmission spectrum has random noise with a Cauchy distribution.

The second noise problem arises because of nonuniform spectral brightness of the source, or because of nonuniform instrument transmission efficiency. The noise in the resulting transmission spectrum often varies significantly in amplitude over the spectral region under study. Often, the ends are much noisier than the center.

Faced with a noisy transmission spectrum, the impulse to smooth is strong. Here is the problem: should one smooth the sample and reference spectra separately, smooth the transmission spectrum, or do something else? We recommend something else.

RazorDivide is the Maximum Likelihood solution to this problem. It provides an *estimate of the noiseless transmission spectrum* which would result from averaging many, many noisy spectra such as the one at hand.

The technique was described in a paper presented at the 1990 Pittsburgh Conference, and is described in Section 3.4.

3.2 `rzrdiv`

`rzrdiv` is a rigorous Maximum Likelihood solution to the problem of ‘smoothing’ a spectrum produced by dividing one noisy spectrum by another noisy spectrum, in that it **yields**

a statistical estimate of the ratio spectrum you would obtain if each of its components were noise-free.

The required user input is:

- The unnormalized sample spectrum.
- The reference spectrum.
- A smooth peakshape characteristic of the narrowest feature of interest present in the data. It is not critical that the user choose an exact peakshape for **rzrdiv**.

Processing notes:

- The resultant smooth transmittance spectrum is constrained to be between 0 and 1.

Programming notes:

- This algorithm is a space hog. It requires six! fill-size arrays, **s**, **r**, **trans**, **u**, **v** and **w**.
Usually one is able to trade memory space for processing time in algorithm development. We were able to do neither here. The **rzrdiv** algorithm uses a lot of array space and a lot of time. Furthermore, it is slow to converge, requiring at least fifty iterations.
- If the noise is not severe, then we recommend the use of **rzresm** separately on sample and reference spectra, before division, as a quicker alternative.
- For the first call, set **iter** = 0. **rzrdiv** will then maintain **iter**.
- The particular method we have used to estimate the smooth ratio spectrum is very slow to converge, especially when the absorbances are small. We continue to work on it..

```
long rzrdiv(float s[ ], float r[ ], long n2,
            float shape[ ], long nl2, float u[ ], float v[ ],
            float w[ ], float trans[ ], long *n, long *newpk,
            long iter, long *nfw hm, double *sigma)
```

Input arrays which must be filled:

s, filled between **0** and **n2**, length **n**
r, filled between **0** and **n2**, length **n**
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:

u, length **n**
v, length **n**
w, length **n**
trans, length **n**

Input variables: **n2**, **nl2**, **n**, **newpk**, **iter**

n2 is the index of the last data value in **s** and **r**.

nl2 is the last position of data in **shape**.

n is the size of arrays **s**, **r**, **u**, **v**, **w** and **trans**.

iter, the iteration number, must be = 0 for the first call.

newpk indicates whether or not **shape** is a new peakshape.

If **newpk** = 1, **shape** will be processed.

Output arrays:

r, filled between **0** and **n2**

Output variables:

n = amount of array space used

NOTE: if **n** is negative, $\text{abs}(n)$ = amount of array space needed (but not available). Operation not successful.

newpk = **n** if **trans** was successfully loaded

nfw hm = full-width-at-half-maximum of peakshape

sigma = RMS noise in **r**

Function return values:

rzrdiv = 0 if successful

If **rzrdiv** < 0, error occurred

Use **rzrerr** (page 174) to obtain error text

Description of variables

s is the *input unnormalized sample data array*. It must be filled between data points **0** and **n2**, and it WILL be altered outside this range by the function, so you may wish to retain a copy before calling **rzrdiv**.

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

r is the *input reference data array*. It will be processed between data points **0** and **n2**, and zeroed outside this range.

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

On *output*, **r** will contain the *smooth transmission, or ratio array*, between data points **0** and **n2**. **r** may be displayed at the end of each iteration, if desired.

n2 is *input* as the *last location* of data in the **s** and **r** arrays which are to be ratioed.

shape is an *input* array which holds the *peakshape of the narrowest spectral feature* in **s** which is of interest to the user. The relevant peakshape is located between data points **0** and **n12** in **shape**.

n12 is *input* and the *index of the last data point of the peakshape* in **shape**. We recommend that **n12+1** be at least $6*nfwhm$, and that the peak be approximately centered in the $(0,n12)$ interval.

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

v is a *work array of size n* which will be used for computations. **v** 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 newly loaded by **rzrddiv**. 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**, **w**, and **trans** arrays.

The function **rzsize** 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 **rzrdiv** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you wish 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 **rzrdiv** 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 **rzrdiv** will be **rzrdiv** = -2.

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

iter is an *input index* for the iteration loop. Set **iter** = 0 for the initial call only. The algorithm distinguishes between **iter** = 0 and **iter** > 0. **rzrdiv** will maintain **iter**.

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

3.3 Example using rzrdiv

The figure shown in Section 3.4 was produced using **rzrdiv**. You can create your own version using **HANDLE.EXE** and the data files:

Unnormalized sample spectrum file: SPEC5

Reference spectrum file: REF5

Peakshape file: PEAK5

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): DIV

```

RazorDivide gives Maximum Likelihood estimate of the ratio of two spectra.

You will need an unnormalized sample spectrum, a reference spectrum, and a peakshape (bandshape).

```
Enter name of unnormalized sample spectrum (Try SPEC5): SPEC5
```

```
Enter name of reference spectrum (Try REF5): REF5
```

```
Enter name of peakshape file (see manual) (Try PEAK5): PEAK5
```

```
Entering RZRDIV with iter=0. Wait for setup...
```

```
At iter 1 the RMS noise is 0.0772
```

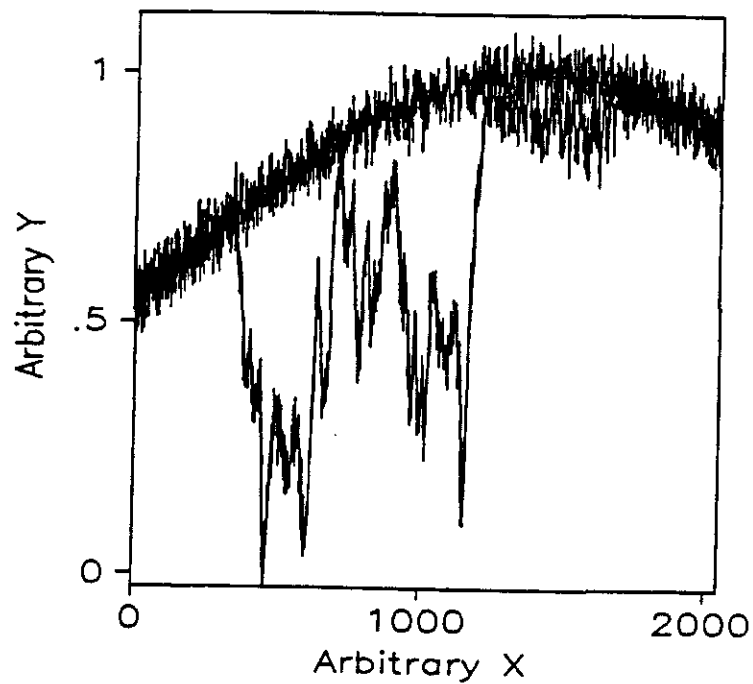
```
.....
.....
```

3.3. EXAMPLE USING RZRDIV

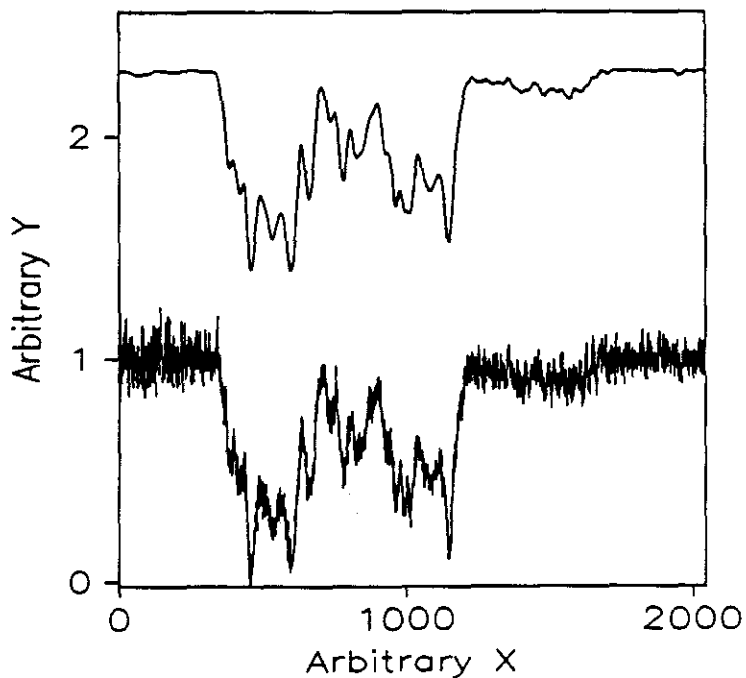
37

At iter 25 the RMS noise is 0.05555
More iterations? Enter the additional number required [0]:
The FWHM of the peakshape is 19.
The size of the array space used was 4096
RESULT MAY BE SAVED TO A FILE

The figure below shows the sample, and the reference.



The second figure shows the transmission spectrum produced by direct division, and the one produced by *rzrdiv*. *rzrdiv* is slow to converge. We let it run 25 iterations. If you do the same, and then overlay the results on the transmission file, you will see that it has not yet converged in the strong absorbance regions. In our experience, it takes 50 or more iterations for convergence. However, *rzrdiv* provides superior noise suppression, as you can see looking at the ends of this spectrum. Sometimes, when the data are very noisy, there is no better way.



3.4 Reducing Noise in Transmission Spectra

(This is a summary of a paper given at the 1990 Pittsburgh Conference.)

Absorption spectroscopy begins with two spectra,

Sample + noise

Reference + noise

Their ratio gives a transmission spectrum,

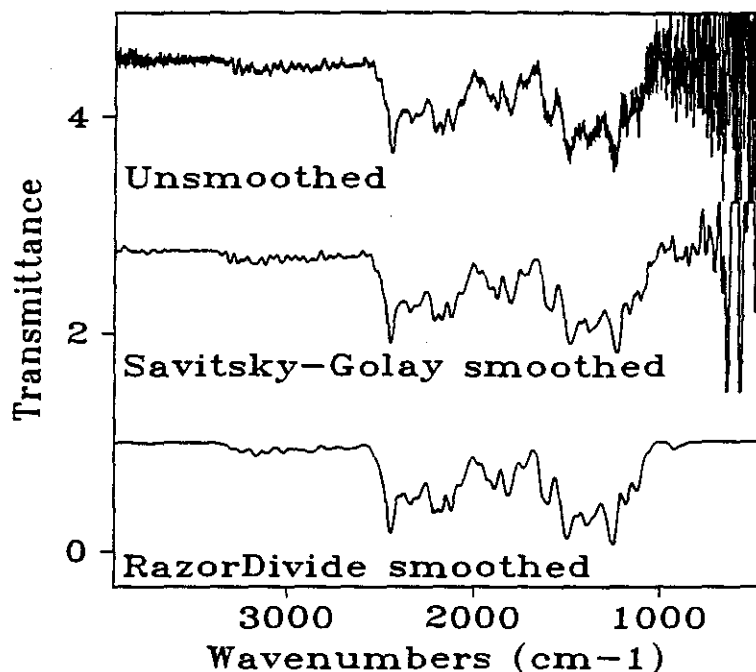
$$T_j = \text{Sample} + \text{noise} / \text{Reference} + \text{noise},$$

where the subscript j is used to remind us that, due to the presence of noise, this particular ratio is just one of the many possible resultant transmission spectra which could have occurred.

The ratio is usually very noisy, especially at the ends of the spectrum where both the sample and the reference have low fluxes. The usual solution is to smooth the transmission spectrum T . However, ordinary smoothing methods are usually not applicable due to the character of the noise. Unfortunately, the noise is not constant, nor even well-behaved! Also, the apparent transmission after smoothing may exceed 100%.

If the sample noise and the reference noise come from Normal probability distributions, then the transmission noise will have a Cauchy distribution. The Cauchy distribution is the origin of the large noise spikes often seen in transmission spectra at the ends of the

observing band. The problem is illustrated in the unsmoothed spectrum below.



Faced with such a noisy transmission spectrum, the impulse to smooth is strong. But should one smooth the sample and reference spectra separately, smooth the transmission spectrum, or do something else? We recommend something else.

One really wants the best possible estimate of a noiseless transmission spectrum, *i.e.*, the spectrum which would result from averaging many, many noisy spectra such as the one at hand. The Maximum Likelihood method provides this estimate. It finds the MOST LIKELY transmission spectrum T , *i.e.* the transmission that would result from averaging many T_j .

Maximum Likelihood smoothing derives its power from the chemist's a priori knowledge about the data. Any chemist or spectroscopist would be able to smooth the data shown above, because his eye tells him the approximate width of true absorption features. Yet each might choose a different smoothing method, or different parameters. Maximum Likelihood is the analytical tool for changing 'smooth by eye' into 'optimum smooth', so that all users obtain the best possible result. In this case, the chemist's intuitive knowledge of peak widths is replaced by one of these statements: the peakshapes are determined by the known instrumental resolution, or the peakshapes are determined by known sample bandshapes.

We have solved the appropriate Maximum Likelihood equations for the transmission problem, incorporating the following a priori knowledge:

- Transmissions are ≤ 1 (i.e., sample absorbances are ≥ 0).
- The shapes of isolated single features in the transmission spectrum are known. These shapes are determined by either
 - Instrumental smearing
 - Intrinsic bandshapes in the sample

The Maximum Likelihood prescription for Cauchy noise distributions is this: To find the best estimate of \mathbf{T} , maximize the probability \mathbf{P} of obtaining the given transmission spectrum $\mathbf{T}_j = \mathbf{S}_j/\mathbf{R}_j$,

$$\mathbf{P}(\mathbf{T}_j) = \frac{1}{1 + \frac{4}{\mathbf{N}_t^2}(\mathbf{T}_j - \mathbf{T})^2},$$

where \mathbf{S}_j is the sample absorption spectrum, and \mathbf{R}_j is the reference spectrum. \mathbf{T} is the transmission spectrum which would result from long-term averaging of many \mathbf{T}_j .

\mathbf{N}_t is the width of the Cauchy noise distribution. \mathbf{N}_t will be a function of the noise \mathbf{N}_s in the sample spectrum \mathbf{S}_j , the noise \mathbf{N}_r in the reference spectrum \mathbf{R}_j , and \mathbf{S} and \mathbf{R} . Thus, $\mathbf{N}_t = \mathbf{N}_t(\mathbf{N}_s, \mathbf{N}_r, \mathbf{S}, \mathbf{R})$. Note that \mathbf{S} , \mathbf{R} , \mathbf{T}_j , \mathbf{T} , \mathbf{N}_s , \mathbf{N}_r , and \mathbf{N}_t are all functions of frequency.

When the peakshapes are determined by sample bandshapes \mathbf{b}_{ik} , then \mathbf{T} is constrained by

$$\mathbf{T} = \exp(-\text{shape}_{ik} * \mathbf{o}(\nu_k)).$$

The probability of obtaining the particular transmission spectrum \mathbf{T}_j then becomes

$$\mathbf{P}(\mathbf{T}_j(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)) = \prod_{i=1}^n \frac{1}{1 + \frac{4}{\mathbf{N}_t^2}(\mathbf{T}_j(\mathbf{x}_i) - \exp(-\mathbf{b}_{ik} * \mathbf{o}(\mathbf{x}_k)))^2}.$$

Maximum Likelihood asks and answers this question:

Of all the possible sets $\{\mathbf{o}\}$, which one maximizes \mathbf{P} ?

The Maximum Likelihood solution is then an estimate of the transmission spectrum \mathbf{T} which would result from long-term averaging.

The solution presented by the RazorDivide algorithm is displayed in the previous figure. The figure shows (top) the original single transmission spectrum \mathbf{T}_j , (middle) Savitsky-Golay smoothing of \mathbf{T}_j , and (bottom) our Maximum Likelihood estimate of the

“smoothed” spectrum **T**. The Maximum Likelihood estimate does not have transmissions $> 100\%$. Further, it *accurately!* finds no absorption below 800 cm^{-1} . (The synthetic sample has no absorption at wavenumbers $\leq 1000\text{ cm}^{-1}$ nor $\geq 3400\text{ cm}^{-1}$).

Maximum Likelihood smoothing outperforms other methods in tough situations, where the signal/noise ratio is low. Furthermore, if the a priori knowledge included in the solution is accurate and complete, then we have obtained the **best possible estimate of the ‘true’ transmission T**.

The benefits of using Maximum Likelihood for obtaining smooth transmission spectra are (1) superior performance in low signal/noise situations, (2) the user does not have to select an arbitrary smoothing procedure, (3) Maximum Likelihood gives the optimum answer, the first time, and (4) all users obtain the same result.

Chapter 4

RazorSharp — rzrash/rzrdec/rzrluc

4.1 Resolution Enhancement without Artifacts

RazorSharp is a collection of Maximum Likelihood and Maximum Entropy/Bayesian methods which enhance resolution, and so can separate overlapping peaks. These methods can increase resolution by factors of two to five, depending upon the signal/noise ratio in the data, and depending on the peak shapes. **RazorSharp** resolution enhancement techniques are proper whenever:

- The spectrum, or chromatogram, in the absence of noise, would have no negative intensities.
- The peaks have been broadened, either by intrinsic physical processes, or by an instrument, and the shape of an isolated broadened peak is known.
- Any baseline, drift, or offset has been removed.
- The noise statistics are either Normal or Poisson.

RazorSharp methods are based upon Maximum Likelihood, Maximum Entropy, and Bayesian principles, and are superior to standard linear “deconvolution” methods in the following ways:

- Do not produce negative artifacts.
- Do not require a high signal/noise ratio.
- Do not produce strong “sidelobes” which mask weak peaks.
- Eliminate requirements to specify a physical parameters, such as a filter shape in the Fourier domain.

RazorSharp attempts to answer the question, “What is the *most likely* enhanced sample spectrum which could have produced the observed data, given the *a priori* knowledge of peak shapes and inherent noise?” The answer will be the same for all users, because the *a priori* information represents physical knowledge about the experiment which can be agreed upon in advance.

4.2 rzrash — RazorASharp

rzrash is the proper resolution enhancement algorithm to use when the peaks in the data set are upright, positive, and unbounded from above, and when the noise is Poisson.

rzrash provides a Maximum Likelihood estimate of the noise-free object spectrum which has been convolved with a known peakshape function to produce the observed absorbance, emission, or counting spectrum.

Required user input:

- Data set in which any **baseline or offset has been properly removed**. Note that **rzrash** is designed for use on data with positive values only!
- Select a peakshape which represents the peaks in the data set. It is *very, very important* that the chosen peakshape be an accurate representation of the true shapes of peaks in the data.

Processing notes:

- The solution is constrained to be positive.
rzrash is designed to work only with positive spectra from which any baseline offset has been previously removed. Remove any baseline or offset! If a baseline shift is not removed, then artifacts *will* be generated.

Programming notes:

- The programmer will need to provide four full-size arrays, **ydata**, **yout**, **w**, and **trans**, for processing. **ydata** *will* be altered outside the range (0,n2).
- Call **rzrash** with **iter** = 0 for the first iteration only. The function will then maintain **iter** for you. Most data files require 15 to 25 iterations.

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

Input arrays which must be filled:

ydata, filled between 0 and **n2**, length **n**

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**

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**, **w** and **trans**

newpk indicates whether or not **shape** is a new peakshape

sigma = RMS noise in **ydata** (optional).

Output arrays:

yout, filled between 0 and **n2**

Output variables:

n = amount of array space used

NOTE: if **n** is negative, $\text{abs}(n)$ = amount of array space needed (but not available). Operation not successful.

newpk = **n** if **trans** was properly loaded.

nfwhm = full-width-at-half-maximum of peakshape

chisq = $((\text{ydata} - \text{yout} - \text{convolved-with-shape}) / \text{sigma})^2$.

sigma = RMS noise in **ydata**.

Function return values:

rzrash = 0 if iteration was successful

If **rzrash** < 0, error occurred

Use **rzrerr** (page 174) to obtain error text

Description of variables

ydata on *input* is the *raw data array*. The data of interest is contained in the range (0, **n2**). **ydata** will be altered outside this range.

On *output*, it will be the *smoothed data array*.

ydata must have a minimum size equal to the smallest power of two larger than $(\mathbf{n2} + 1 + 3 * \mathbf{nfwhm})$. 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 the **shape** array. The minimum size of the **shape** array is **nl2+1**. **nl2** must always be less than **n**.

nl2 is *input* as the *index of the last data point of the peakshape* in **shape**. We recommend that **nl2+1** be at least $6 \times \text{nfwhm}$, and that the peak be approximately centered in the **0,nl2** interval.

yout is an *array of size n*. On *output*, **yout** will be the *resolution-enhanced data array*. **yout** is available for display at the end of each iteration.

yout must have a minimum size equal to the smallest power of two larger than $(\text{n2}+1+3 \times \text{nfwhm})$. 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 $(\text{n2}+1+3 \times \text{nfwhm})$. 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 thus **trans** is newly loaded by **rzrash**. 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**, **w**, and **trans** arrays.

The function **rsizn** 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 = rsizn(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 $(\text{n2}+1+3 \times \text{nfwhm})$. You may wish to calculate **n** in an alternate fashion. See Chapter 11.

NOTE: When **rzrash** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you wish 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 **rzrash** 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 **rzrash** will be **rzrash** = -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, and will automatically update the value of **iter**.

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

chisq is *output* as the *standard deviation (root-mean-square) of the difference between the observed data ydata and the result spectrum yout convolved with shape, normalized to the RMS noise sigma*. It is a measure of the convergence of the algorithm. It may be displayed at the end of each iteration.

sigma is either/both an *input* and an *output* variable. It is the *standard deviation (root-mean-square)* of the noise.

When the RMS noise in the **ydata** array is **known**, it should be *input* in **sigma**.

When the RMS noise is not known, set **sigma** = 0.0, as a signal to the function to auto-calculate **sigma**.

4.3 Example using rzrash

Benzene is a favorite for testing the resolution of a slit spectrometer. One of the spectra in the figure below was taken with a 2 nanometer slit setting. At the same time, the operator scanned one of the narrow lines of his deuterium lamp, providing us with the spectrum we give you in the file PEAK1.

We passed these files through the HANDLE program contained on your disk, using RazorASharp (Command ASH). The result is shown.

Data file: SPEC1

Peakshape: PEAK1

Using HANDLE:

```

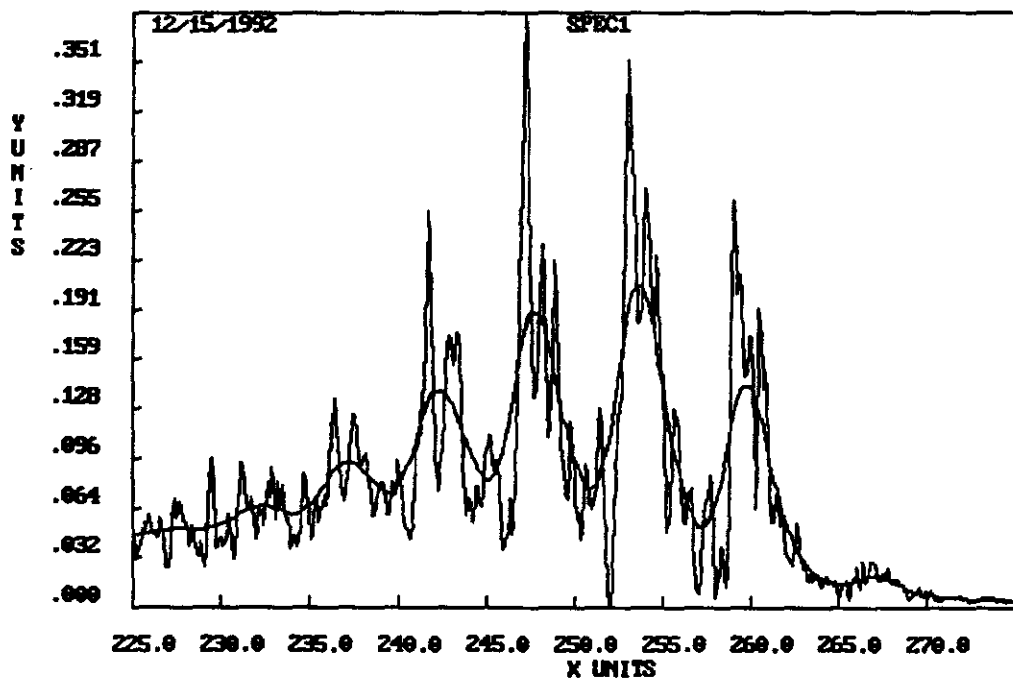
RAZOR LIBRARY for Spectral Analysis -; There is only one best way!
Maximum Likelihood (ML) , Maximum Entropy (ME) , and Bayesian processing.
ESM=EntropySmooth. Smooths Normal (thermal/gaussian) noise. ME
PSM=PoissonSmooth. Smooths Poisson (counting) noise. ML.
NSM=NormalSmooth. Smooths Normal noise. 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) : ASH
Enter name of unnormalized sample spectrum (Try SPEC1) : SPEC1
Enter name of peakshape file (Try SPEC1) : PEAK1

Entering RZRASH with iter = 0. Wait for setup...
At iter 1, RMS noise = .000171, Chisq = 159.40
.....
.....
At iter 15, RMS noise = .000171, Chisq = 2.017
More iterations? Enter the additional number required [0] : 0
ASH: Final RMS noise = .000171, Chisq = 2.017
The FWHM of the peakshape is 20.

```

RESULT MAY BE SAVED TO A FILE

ASH = RazorSharp: Iter= 15, RMS noise= .323130E-03, Chisq= .115803E+01
More iterations? Enter the additional number required: [0]



4.4 rzddec — RazorDEConvolve

rzddec is the proper resolution enhancement algorithm to use when the peaks in the data set are upright, positive, and unbounded from above, and when the noise is Normal.

rzddec provides a Maximum Entropy estimate of the noise-free object spectrum which has been convolved with a known peakshape function to produce the observed absorbance, or emission, spectrum.

Required user input:

- Data set in which any **baseline or offset has been properly removed**. Note that **rzddec** is designed for use on data with positive values only!
- Select a peakshape which represents the peaks in the data set. It is *very, very important* that the chosen peakshape be an accurate representation of the true shapes of peaks in the data. The correct peakshape is more important for **rzddec** than for any other function of Razor Library, because the underlying algorithm of **rzddec** is more powerful than any other, and thus it is more sensitive to nuances in the peakshape.

Processing notes:

- The solution is constrained to be positive.
rzddec is designed to work only with positive spectra from which any baseline offset has been previously removed. Remove any baseline or offset! If a baseline shift is not removed, then artifacts *will* be generated.

Programming notes:

- The programmer will need to provide four full-size processing arrays, **yout**, **w**, **x**, and **trans**, as well as the data array **ydata**. **ydata** will NOT be altered by **rzddec**.
- Call **rzddec** with **iter** = 0 for the first iteration only. The function will then maintain **iter** for you. Most data files require 100 to 200 iterations.
- **rzddec** contains the most powerful processing algorithm in Razor Library. The algorithm is so powerful that it eventually reaches the limits of double precision arithmetic. When this happens, **rzddec** has converged within the limits available. It will send back a return value of -10, indicating that further iterations would give no improvement. You may look for this return value as a natural stopping point, and use the output **yout** with confidence!

```

long rzrdec(float ydata[ ], float prior[ ], long n2, float shape[ ], long nl2,
  float yout[ ], float v[ ], float w[ ], float x[], float trans[ ], long *n, long *newpk,
int
  pflag, long *iter, double things[], long *nfwhm, double *chisq, double *sigma)

```

Input arrays which must be filled:

ydata, filled between 0 and **n2**, length **n2 + 1**
prior, optionally filled between 0 and **n2**, length 1, or **n2 + 1**
 NOTE: If **pflag** = 0, **prior** will not be used.
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**
v, length **n**
w, length **n**
x, length **n**
trans, length **n**
things, length = 10

Input variables: **n2**, **nl2**, **n**, **newpk**, **pflag**, **iter**, **things[10]**

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**, **w** and **trans**
newpk indicates whether or not **shape** is a new peakshape
pflag indicates whether or not **prior** contains information
iter must be set to 0 for the first iteration
things[0], etc should be set to 0 for standard operation
sigma = RMS noise in **ydata** (optional).

Output arrays:

yout, filled between 0 and **n2**

Output variables:

n = amount of array space used
 NOTE: if **n** is negative, $\text{abs}(n)$ = amount of array space
 needed (but not available). Operation not successful.
newpk = **n** if **trans** was properly loaded.
iter will be updated to show the next iteration number
nfwhm = full-width-at-half-maximum of peakshape
chisq = $((\text{ydata} - \text{yout-convolved-with-shape})/\text{sigma})^2$.
sigma = RMS noise in **ydata**.
things[4] = Entropy of resolved configuration in **yout**

Function return values:

rzrdec = 0 if iteration was successful

= -10 when it reaches the limits of double precision arithmetic
 If **rzrdec** < 0, and != -10, an error occurred
 Use **rzrerr** (page 174) to obtain error text

Description of variables

ydata on *input* is the *raw data array*. The data of interest is contained in the range (0, **n2**). **ydata** will NOT be altered outside this range.

prior on *input* is the *your best (biased) estimate of the output array*.

When **pflag** = 0, **rzrdec** uses a flat prior, and the **prior** array is ignored.

When **pflag** = 1, the prior is a smoothed version of the data. In this case, you only need to provide an array of size **n2**. **rzrdec** will fill it and maintain it.

When **pflag** = 2, **rzrdec** will read the array **prior** to find *your* prior estimate of the deconvolved result.

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 **n12** in the **shape** array. The minimum size of the **shape** array is **n12+1**. **n12** must always be less than **n**.

n12 is *input* as the *index of the last data point of the peakshape* in **shape**. We recommend that **n12+1** be at least $6 \cdot \mathbf{nfwhm}$, and that the peak be approximately centered in the 0,**n12** interval.

yout is an *array of size n*. On *output*, **yout** will be the *resolution-enhanced data array*. **yout** is available for display at the end of each iteration.

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

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

x is a *work array of size n* which will be used for computations. **x** must have a minimum size equal to the smallest power of two larger than $(\mathbf{n2}+1+3 \cdot \mathbf{nfwhm})$. 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 thus **trans** is newly loaded by **rzrdec**. 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**, **v**, **w**, and **trans** arrays.

The function **rsizn** 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 = rsizn(n2,shape,n12)
```

On *output*, **n** is the amount of *space used for the Fourier transforms* in the **yout**, **v**, **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 **rzrdec** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you wish 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 **rzrdec** 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 **rzrdec**

will be **rzrdec** = -2.

pflag is an *input* flag that tells **rzrdec** your *a priori* estimate of the true result. When **pflag** = 0, **rzrdec** uses a flat prior, and the **prior** array is ignored. When **pflag** = 1, the prior is a smoothed version of the data. The **prior** array *will* be used; **rzrdec** will fill it and maintain it. When **pflag** =2, **rzrdec** will read the array **prior** to find *your* prior estimate of the deconvolved result.

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, and will automatically update the value of **iter**.

things is a work array which holds parameters that must be saved between iterations. On *input*, before the first iteration, set **things** = 0.0, for standard operation.

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

chisq is *output* as the *standard deviation (root-mean-square) of the difference between the observed data ydata and the result spectrum yout convolved with shape, normalized to the RMS noise sigma*. It is a measure of the convergence of the algorithm. It may be displayed at the end of each iteration.

sigma is either/both an *input* and an *output* variable. It is the *standard deviation (root-mean-square)* of the noise.

When the RMS noise in the **ydata** array is **known**, it should be *input* in **sigma**.

When the RMS noise is not known, set **sigma** = 0.0, as a signal to the function to auto-calculate **sigma**.

4.5 Example using rzddec

Benzene is a favorite for testing the resolution of a slit spectrometer. One of the spectra in the figure below was taken with a 2 nanometer slit setting. At the same time, the operator scanned one of the narrow lines of his deuterium lamp, providing us with the spectrum we give you in the file PEAK1.

We passed these files through the HANDLE program contained on your disk, using RazorDeconvolve (Command DEC). The result is shown.

Data file: SPEC1

Peakshape: PEAK1

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): DEC
Enter name of unnormalized sample spectrum (Try SPEC1): SPEC1
Enter name of peakshape file (Try SPEC1): PEAK1

```

RZRDEC is Bayesian, and uses an a priori spectrum.

Sdt pflag=0 for uniform prior. [Default]

Set pflag=1 to use smoothed data as prior.

Choose pflag (Use 0 if not sure): 0

Entering RZRDEC with iter = 0. Wait for setup...

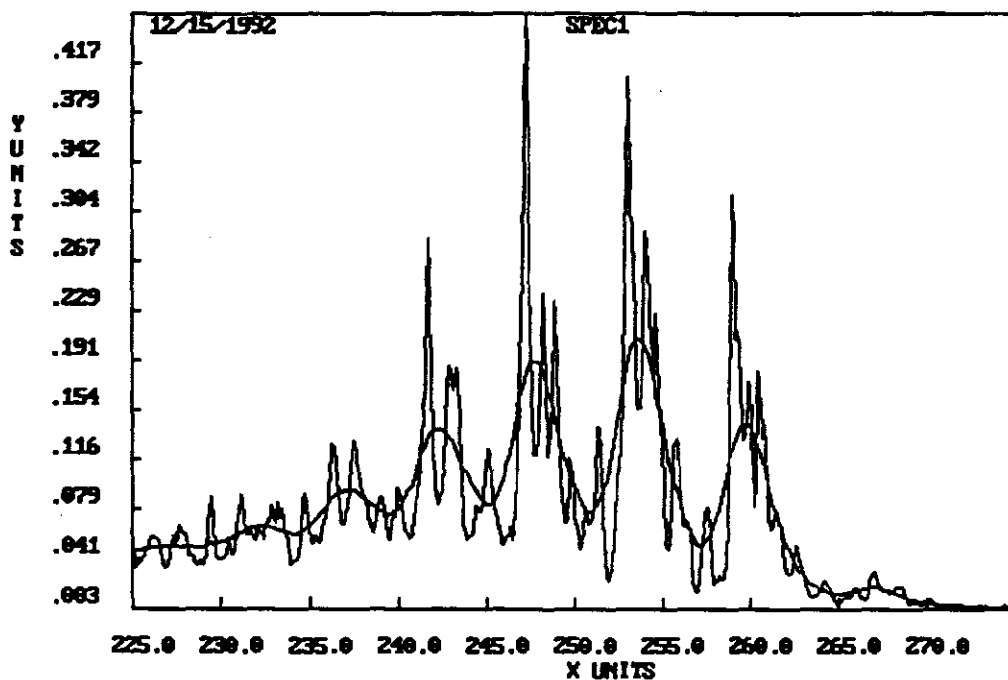
RZRDEC Iter= 1, RMS= .000171, Chisq= 252681, Mean Entropy= -2.625

RZRDEC Iter= 2, RMS= .000171, Chisq= 13037.2, Mean Entropy= -.0834

4.5. EXAMPLE USING RZRDEC

.....
.....
RZRDEC Iter= 97, RMS= .000171, Chisq = 15.96, Entropy=-4.519
RZRDEC has finished!
The FWHM of the peakshape is 20.
The size of array space used was 1024
RESULT MAY BE SAVED TO A FILE

DEC = RazarDeconvolve: At iter 25 the reduced chi-squared is .566389E+01
More iterations? Enter the additional number required: [0]



4.6 rzrluc — RazorLUCy

rzrluc is the proper resolution enhancement algorithm to use when the peaks in the data set are upright, positive, and unbounded from above, and when the noise is Poisson.

rzrluc provides a Maximum Likelihood estimate of the noise-free object spectrum which has been convolved with a known peakshape function to produce the observed absorbance, emission, or counting spectrum.

Required user input:

- Data set in which any **baseline or offset has been properly removed**. Note that **rzrluc** is designed for use on data with positive values only!
- Select a peakshape which represents the peaks in the data set. It is *very, very important* that the chosen peakshape be an accurate representation of the true shapes of peaks in the data.

Processing notes:

- The solution is constrained to be positive.
rzrluc is designed to work only with positive spectra from which any baseline offset has been previously removed. Remove any baseline or offset! If a baseline shift is not removed, then artifacts *will* be generated.

Programming notes:

- The programmer will need to provide four full-size arrays, **ydata**, **yout**, **w**, and **trans**, for processing. **ydata** *will* be altered outside the range (0,n2).
- Call **rzrluc** with **iter** = 0 for the first iteration only. The function will then maintain **iter** for you. Most data files require 15 to 25 iterations.

```
long rzrluc(float ydata[ ], long n2, float shape[ ], long nl2,
           float yout[ ], float w[ ], float trans[ ], long *n, long *newpk,
           long *iter, long *nfw hm, double *chisq, double *sigma)
```

Input arrays which must be filled:

ydata, filled between 0 and **n2**, length **n**

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**

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**, **w** and **trans**

newpk indicates whether or not **shape** is a new peakshape

sigma = RMS noise in **ydata** (optional).

Output arrays:

yout, filled between 0 and **n2**

Output variables:

n = amount of array space used

NOTE: if **n** is negative, $\text{abs}(n)$ = amount of array space needed (but not available). Operation not successful.

newpk = **n** if **trans** was properly loaded.

nfw hm = full-width-at-half-maximum of peakshape

chisq = $((\text{ydata} - \text{yout-convolved-with-shape})/\text{sigma})^2$.

sigma = RMS noise in **ydata**.

Function return values:

rzrluc = 0 if iteration was successful

If **rzrluc** < 0, error occurred

Use **rzrerr** (page 174) to obtain error text

Description of variables

ydata on *input* is the *raw data array*. The data of interest is contained in the range (0, **n2**). **ydata** will be altered outside this range.

On *output*, it will be the *smoothed data array*.

ydata must have a minimum size equal to the smallest power of two larger than $(\text{n2}+1+3*\text{nfw hm})$. 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 **n12** in the **shape** array. The minimum size of the **shape** array is **n12+1**. **n12** must always be less than **n**.

n12 is *input* as the *index of the last data point of the peakshape* in **shape**. We recommend that **n12+1** be at least $6 \times \text{nfwhm}$, and that the peak be approximately centered in the **0,n12** interval.

yout is an *array of size n*. On *output*, **yout** will be the *resolution-enhanced data array*. **yout** is available for display at the end of each iteration.

yout must have a minimum size equal to the smallest power of two larger than $(\mathbf{n2}+1+3 \times \text{nfwhm})$. 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 $(\mathbf{n2}+1+3 \times \text{nfwhm})$. 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 thus **trans** is newly loaded by **rzrluc**. 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**, **w**, and **trans** arrays.

The function **rzsize** 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 = rzsize(n2,shape,n12)
```

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 $(\mathbf{n2}+1+3 \times \text{nfwhm})$. You may wish to calculate **n** in an alternate fashion. See Chapter 11.

NOTE: When **rzrluc** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you wish 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 **rzrluc** 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 **rzrluc** will be **rzrluc** = -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, and will automatically update the value of **iter**.

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

chisq is *output* as the *standard deviation (root-mean-square) of the difference between the observed data ydata and the result spectrum yout convolved with shape, normalized to the RMS noise sigma*. It is a measure of the convergence of the algorithm. It may be displayed at the end of each iteration.

sigma is either/both an *input* and an *output* variable. It is the *standard deviation (root-mean-square)* of the noise.

When the RMS noise in the **ydata** array is **known**, it should be *input* in **sigma**.

When the RMS noise is not known, set **sigma** = **0.0**, as a signal to the function to auto-calculate **sigma**.

4.7 Example using rzrluc

Data file: SPEC2

Peakshape: PEAK2

Using HANDLE:

```

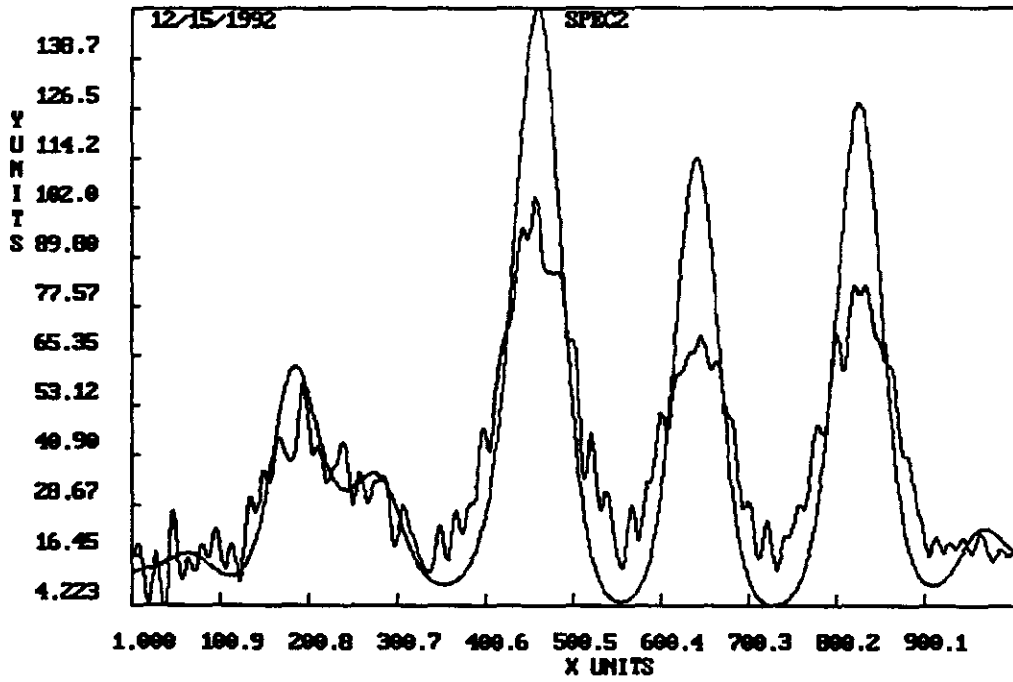
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
ESM=EntropySmooth. Smooths Normal (thermal/gaussian) noise. ME
PSM=PoissonSmooth. Smooths Poisson (counting) noise. ML.
NSM=NormalSmooth. Smooths Normal noise. ML.
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): LUC
Enter name of unnormalized sample spectrum (Try SPEC2): SPEC2
Enter name of peakshape file (Try SPEC2): PEAK2

Entering RZRLUC with iter = 0. Wait for setup...
At iter 1, RMS noise = 3.4088, Chisq = 2.636
.....
.....
At iter 15, RMS noise = 3.4088, Chisq = 1.270
More iterations? Enter the additional number required [0]: 0
The FWHM of the peakshape is 80.
The size of array space used was 1024
RESULT MAY BE SAVED TO A FILE

```

4.7. EXAMPLE USING RZRLUC

More iterations? Enter the additional number required: [0]
Result may be saved to a file



4.8 Statistically Sound Restoration

The functions in RazorSharp are Maximum Likelihood and Maximum Entropy/Bayesian Restoration methods. Spectrum Square specializes in methods of this type, because they are statistically sound and immune to the usual diseases found in linear “deconvolution” techniques. Although Maximum Likelihood and Maximum Entropy restoration is widely used in geophysics and astrophysics, the principles are not generally known to spectroscopists. We will describe them here.

4.9 The Bayesian Principle

Every spectroscopist knows that a spectrometer distorts, even as it reveals, the spectrum produced by a sample. A feature that the chemist suspects is a group of sharp peaks may appear in a spectrum as a single broad asymmetric peak, probably contaminated by noise. This may not surprise the chemist, but also may not help him much. If he can run the sample again with sufficient resolution then he will do so, but if this is not possible then he is faced with a problem of interpretation. From previous experience he may know what a single sharp peak looks like when viewed “through” the spectrometer, and he may then try to guess what the composition of the observed feature must be in order to appear as it does. That is, the spectroscopist will try to decide *by eye* what the most likely input spectrum must have been, to have produced the observed spectrum. It’s a little like trying to fit a straight line to a data set by eye, but a lot harder.

There should be a better way to solve problems of this kind, and there is. It’s called Bayesian Spectral Restoration. Bayesian restoration responds to the spectroscopist’s need by answering the question,

“What is the *most likely* sample behavior that could have resulted in the observed spectrum, given a specific set of known characteristics of the observing system?”.

The answer is then given in a statistically reliable and reproducible manner.

4.10 How Bayesian/Maximum Likelihood/Maximum Entropy Restoration Works

We have asked the question, what is the *most probable* sample spectrum (usually called the object spectrum), consistent with the data at hand, and consistent with a set of known system constraints?

These constraints consist of everything the experimenter knows about the system, including the data set he is trying to interpret. “Everything” may include but is not limited to:

4.10. HOW BAYESIAN/MAXIMUM LIKELIHOOD/MAXIMUM ENTROPY RESTORATION WORKS⁶⁵

1. The spectrometer peakshape function. (What does it do to a single peak?)
2. The signal-to-noise ratio.
3. Whether the signal has an upper and/or a lower bound, and if it does, what these bounds are.
4. The noise. (Is the noise additive or multiplicative? Is it Normal or Poisson? What are the spectral characteristics of the noise?)
5. The total energy, always presumed to be conserved.
6. A priori object spectrum probabilities. (For example, in atomic mass spectrometry, signals can appear only at certain positions.)

The list can go on. Some of the constraints are very well known and have small variability; these are often taken as “known” to simplify the computation. An example is the instrument peakshape function. Other constraints, such as detector noise, are not known except as to type of statistical behavior and mean square magnitude. In practice, the more one knows about the system, the better. Since we are adding knowledge that is not implicit in the data, it is well to add a lot of it, provided it is correct!

We emphasize that appropriate constraints will all be known or ascertainable for a given system. One does not tinker with them to obtain a result one *likes*, any more than one pushes data points around to obtain a least squares fit whose slope and intercept one likes. We want, after all, the most likely result. If it is not pleasing, well, we did the best we could with the data we had!

Maximum Likelihood algorithms are constructed around the following paradigm: The object spectrum is caused by a physical process which is statistically stationary, so that successive samples are short-time approximations to some mean behavior which has a definite limit as the sampling period increases without limit.

We see at once that this assumption cannot be correct. Light sources burn out, the sample evaporates, etc. Nevertheless, we must make the assumption that at least during the time of the experiment, nothing has changed. In particular, nothing in the above list of constraints changes during the data acquisition period.

One then constructs a probability function which assigns a probability to every physically possible outcome of a particular experiment. One finds that for some object choices, the observed data spectrum is very probable; for other arrangements of the object spectrum, the observed data spectrum is highly unlikely. We simply choose that object spectrum which has the highest probability of giving us our observed data set.

There is no magic here, and no way to adjust the outcome. One does not guess, except in the sense that one guesses that 1000 tosses of a fair coin on flat ground will result in most of the trials ending with the coin flat on the ground. (Such a prediction is not really a guess, but the result of a rapid Maximum Likelihood analysis!)

4.11 Equations used by rzrash and rzrdec and rzrluc

The equations needed for Bayesian/Maximum Likelihood/Maximum Entropy spectral restoration are not particularly difficult to set up.

Suppose we have measured a data set $\{y_1, y_2, \dots, y_n\}$. We really want to know the values of the data set $\{o_1, o_2, \dots, o_n\}$, where each y_i is related to o_i by the equation

$$y_i = (\mathbf{o} \otimes \mathbf{s})_i + n_i.$$

The set $\{o_1, o_2, \dots, o_n\}$ is a more highly resolved spectrum, \otimes denotes convolution, s is the peakshape function, and n_i are the noise fluctuations. We will use Maximum Likelihood and Bayesian methods to estimate $\{o_1, o_2, \dots, o_n\}$.

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

$$p(y_i | \mathbf{o}) = \frac{1}{\sqrt{(2\pi)\sigma_i}} \exp\left[-\frac{(y_i - (\mathbf{o} \otimes \mathbf{s})_i)^2}{2\sigma_i^2}\right].$$

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

$$p(y_i | \mathbf{o}) = \frac{(\mathbf{o} \otimes \mathbf{s})_i^{y_i} e^{-(\mathbf{o} \otimes \mathbf{s})_i}}{y_i!}.$$

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

$$p(y_1, \dots, y_n | \mathbf{o}_1, \dots, \mathbf{o}_n) = \prod_{i=1}^n p(y_i | \mathbf{o})$$

For Normal noise, this becomes

$$p(y_1, \dots, y_n | \mathbf{o}_1, \dots, \mathbf{o}_n) = \prod_{i=1}^n \frac{1}{\sqrt{(2\pi)\sigma_i}} \exp\left[-\frac{(y_i - (\mathbf{o} \otimes \mathbf{s})_i)^2}{2\sigma_i^2}\right].$$

For Poisson noise,

$$p(y_1, \dots, y_n | \mathbf{o}_1, \dots, \mathbf{o}_n) = \prod_{i=1}^n \frac{(\mathbf{o} \otimes \mathbf{s})_i^{y_i} e^{-(\mathbf{o} \otimes \mathbf{s})_i}}{y_i!}.$$

4.11.1 Maximum Likelihood Restoration

When we maximize the probability

$$p(y_1, \dots, y_n | \mathbf{o}_1, \dots, \mathbf{o}_n),$$

in order to find the best estimate of $\{o_1, o_2, \dots, o_n\}$, then we are performing **Maximum Likelihood Spectral Restoration**.

