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

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 sotware.

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 **rzrpic** and **rzrbas**. The peaks may be sorted by significance, height, width, or location. **rzpkst** is described on Page 175, and source code for using this utility is given in **handle.for**.

- A new utility **rzdfil** will help you fill the **datmat** input array for **rzrfit**, using the output arrays from **rzrpic** and **rzrbas**. See Page 177, and **handle.for**.

- **rzrpic** and **rzrbas** will automatically search for *negative peaks*, if a negative peak is presented in the **shape** array.

Contents

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 — rzrdiv **31**

3.1 Noise Reduction for Ratio Spectra 31
3.2 **rzrdiv** 31
3.3 Example using **rzrdiv** 36
3.4 Reducing Noise in Transmission Spectra 39

4 RazorSharp — rzrash/rzrdec/rzrluc **43**

4.1 Resolution Enhancement without Artifacts 43
4.2 **rzrash** — 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 — rzrpic **87**
6.1 Accurate Peak-Picking for Merged Peaks . . . 87
6.2 **rzrpic** . . . 88
6.3 Example using **rzrpic** . . . 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 **rzrpic** 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 Pearson7 . . . 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 — rzrnoi **157**
10.1 **rzrnoi** . . . 157
10.2 Example using **rzrnoi** . . . 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 Calc^{tm}, Lab Calc^{tm}, and GRAMS/386^{tm}.

1.2 Principal Razor functions

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

RazorSmooth: Maximum Likelihood estimation of the smooth parent distribution of a noisy data set.

RazorEntropySmooth — **rzresm**
RazorPoissonSmooth — **rzrpsm**
RazorNormalSmooth — **rzrnsm**

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

— **rzrdiv**

RazorSharp: Maximum Likelihood and Maximum Entropy/Bayesian resolution sharpening and enhancement.

RazorA-Sharp — **rzrash**
RazorDeconvolve — **rzrdec**
RazorLucy — **rzrluc**

RazorDerivative: Bayesian Derivatives.

— **rzrdif**

RazorPick: Maximum Likelihood/Bayesian peak picking.

— **rzrpic**

RazorFit: Maximum Likelihood fitting model peaks to data.

— **rzrfit**

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

RazorBase — **rzrbas**

RazorQuickBase — **rzrqba**

RazorEdge — **rzredg**

RazorCut — **rzrcut**

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

— **rzrnoi**

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

1.3 User input and programmer control

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

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

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

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

1.4 Example source code

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

1.5 Source for service functions

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

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

Chapter 2

RazorSmooth — rzresm/rzrpsm/rzrnsm

2.1 Smoothing which Preserves Resolution

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

The **RazorSmooth** functions give:

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

2.2 Which one to use?

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

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

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

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

2.3 rzresm — Razor Entropy Smooth

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

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

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

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

Processing notes:

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

Programmer notes:

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

  The number of arrays may be reduced to 2 by setting the output array **yout** to the input array **ydata**.

- **ydata** *will not* be altered outside the data region **0** - **n2**, unless you elect to do the processing in-place by setting **yout** = **ydata**.

- If you are processing many scans, all of the same length, and using the same peakshape for all, save processing time with this tactic. Call **rzresm** the first time with **newpk** = 1, thereafter with **newpk** unchanged. When **newpk** = 1, all the functions in Razor Library transfer a properly scaled, properly phased, copy of the input **shape** array into the array **trans**, and then perform an FFT on the **trans** array. When **newpk** > 1, the functions ignore **shape**, and use **trans** directly. This saves the time of a Fourier transform on **trans**. (Note that when **newpk** > 1, **shape** can be a dummy array of length 1, since it will not be used.)

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

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

Description of variables

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

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

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

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

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

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

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

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

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

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

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

n = rzsizn(n2,shape,nl2)

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

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

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

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

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

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

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

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

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

2.4 Example using rzresm

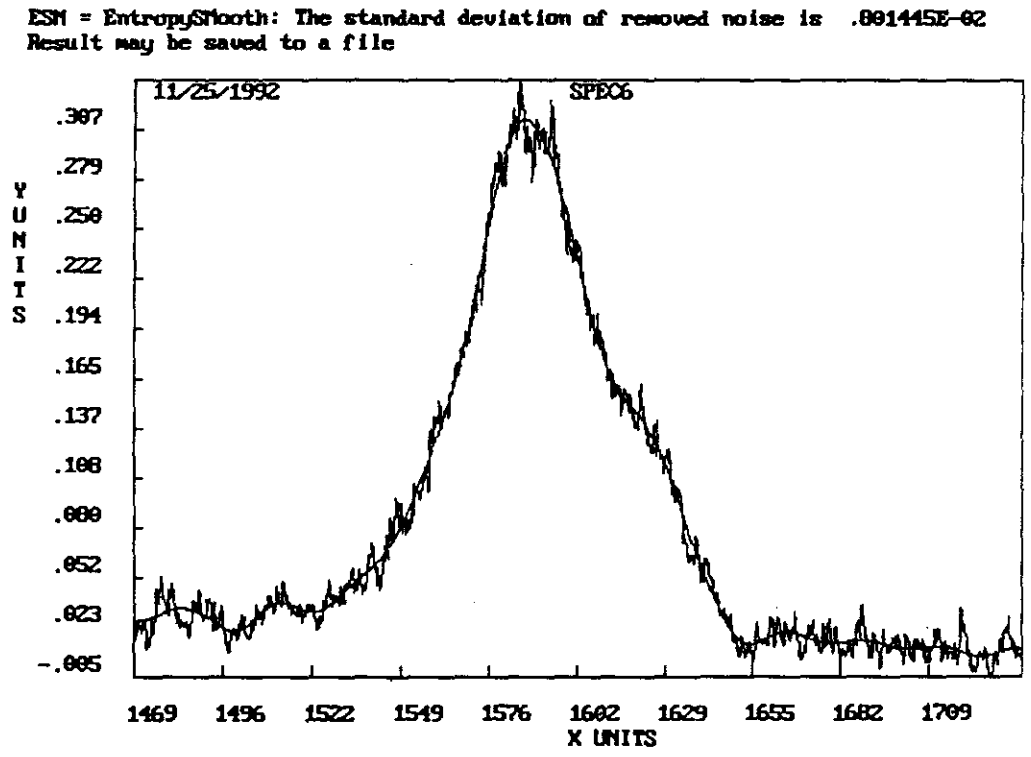
Raman microprobe spectra rarely have enough photons. SPEC6, which is a microprobe spectrum of a carbon thin film, is no exception. We smoothed the spectrum shown below using a Lorentzian, 150 points wide, stored in PEAK6.

Data file: SPEC6

Peakshape file: PEAK6

Using HANDLE:

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

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

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

Press ENTER to return to menu.
```


ESM = EntropySMooth: The standard deviation of removed noise is .001445E-02
Result may be saved to a file
11/25/1992
SPEC6
.307
.279
.250
.222
.194
.165
.137
.108
.080
.052
.023
-.005
Y UNITS
1469
1496
1522
1549
1576
1602
1629
1655
1682
1709
X UNITS

2.5 rzrpsm — Razor Poisson Smooth

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

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

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

Processing notes:

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

Programmer notes:

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

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

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

Description of variables

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

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

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

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

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

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

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

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

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

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

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

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

n = rzsizn(n2,shape,nl2)

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

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

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

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

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

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

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

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

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

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

2.6 Example using rzrpsm

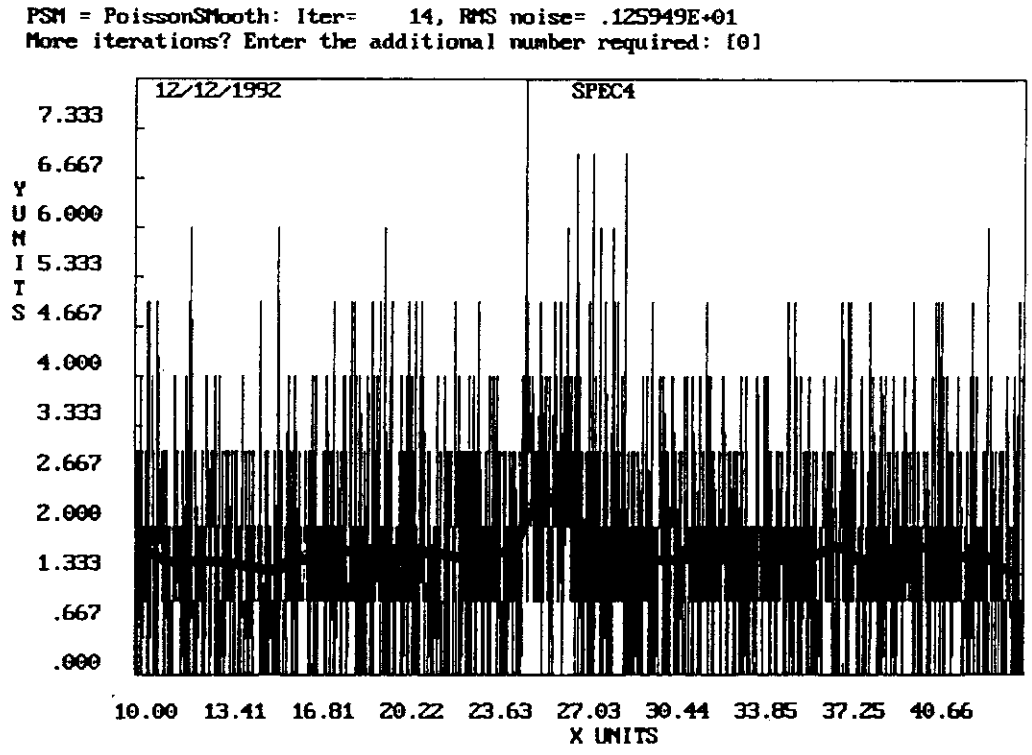
The data file SPEC4 represents radio chromatography, where one is counting nuclear disintegrations in a flow cell, and the background counts are 50% of the total signal. The peakshape is derived from a strong peak in the same cell. The original data had more noise than you will see in the peakshape data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

Data file: SPEC4
Peakshape file: PEAK4

Using HANDLE:

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

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

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

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



2.7 rzrnsm — Razor Normal Smooth

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

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

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

Processing notes:

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

Programmer notes:

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

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

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

Description of variables

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

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

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

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

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

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

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

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

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

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

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

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

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

n = rzsizn(n2,shape,nl2)

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

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

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

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

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

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

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

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

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

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

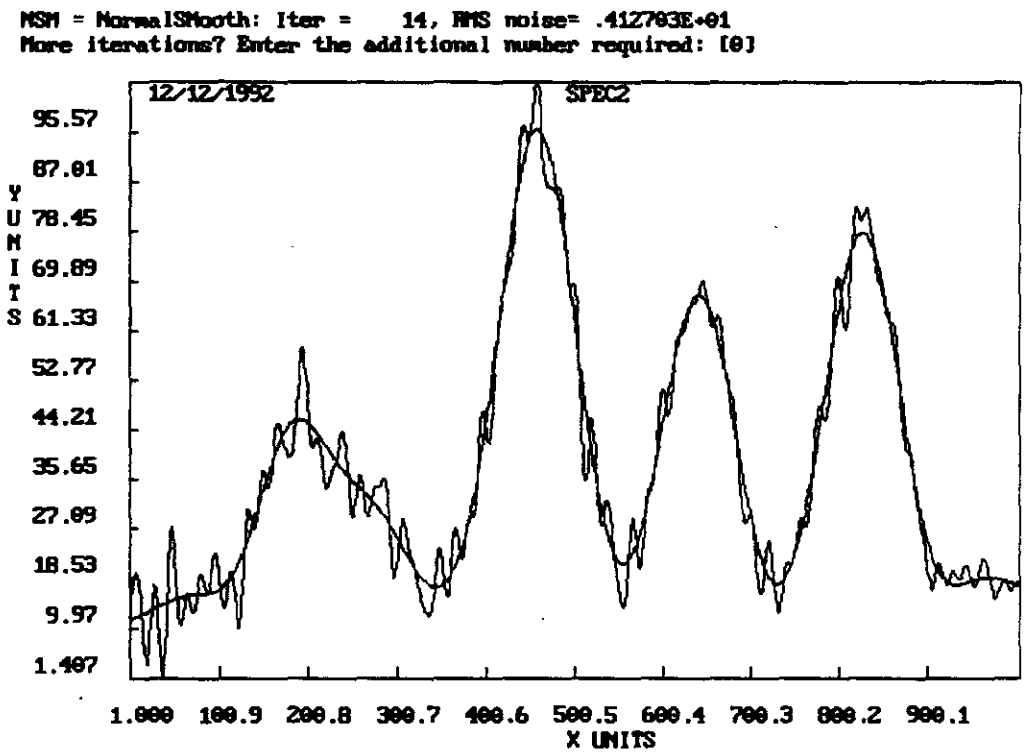
2.8 Example using rzrnsm

Data file: SPEC2

Peakshape file: PEAK2

Using HANDLE:

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

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

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

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


NSM = NormalSMooth: Iter = 14, RMS noise= .412703E+01
More iterations? Enter the additional number required: [0]
12/12/1992
SPEC2
95.57
87.01
78.45
69.89
61.33
52.77
44.21
35.65
27.09
18.53
9.97
1.407
Y UNITS
1.000 100.9 200.8 300.7 400.6 500.5 600.4 700.3 800.2 900.1
X UNITS

2.9 Maximum Likelihood Smoothing — Theory

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

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

2.10 The purpose of a smoothing formula

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

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

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

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

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

2.11 Maximum Likelihood Foundation

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

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

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

2.12 Smoothing Equations

We want to set up an equation which describes the probability of obtaining the data set $\{y_1, y_2, ...y_n\}$, in terms of the parent spectrum $\{z_1, z_2, ...z_n\}$. For a given parent spectrum, the probability $\mathbf{p}$ for the sample $\{y_1, y_2, ...y_n\}$ is determined by the probability distributions for the noise $\{n_1, n_2, ...n_n\}$. $\mathbf{p}$ 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 $\{y_1, y_2, ...y_n\}$. From a parent spectrum $\{z_1, z_2, ...z_n\}$, we have drawn a data set $\{y_1, y_2, ...y_n\}$ containing data points $y_1, y_2, ...y_n$, where $y_i = z_i + n_i$. The n_i are the noise values.

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

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

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

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

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

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

For Normal noise, this becomes

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

For Poisson noise,

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

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

We now proceed in one of two ways:

(1) We maximize the probability

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

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

(2) We maximize the probability

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

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

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

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

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

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

.

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

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

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

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

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

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

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

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

.

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

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

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

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

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

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

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

2.14 Limitations of rzrpsm and rzrnsm

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

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

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

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 *nfwhm, 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, abs(n) = amount of array space needed (but not available). Operation not successful.
newpk = n if **trans** was successfully loaded
nfwhm = 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 **nl2** in **shape**.

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

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

n = rzsizn(n2,shape,nl2)

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

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

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

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

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

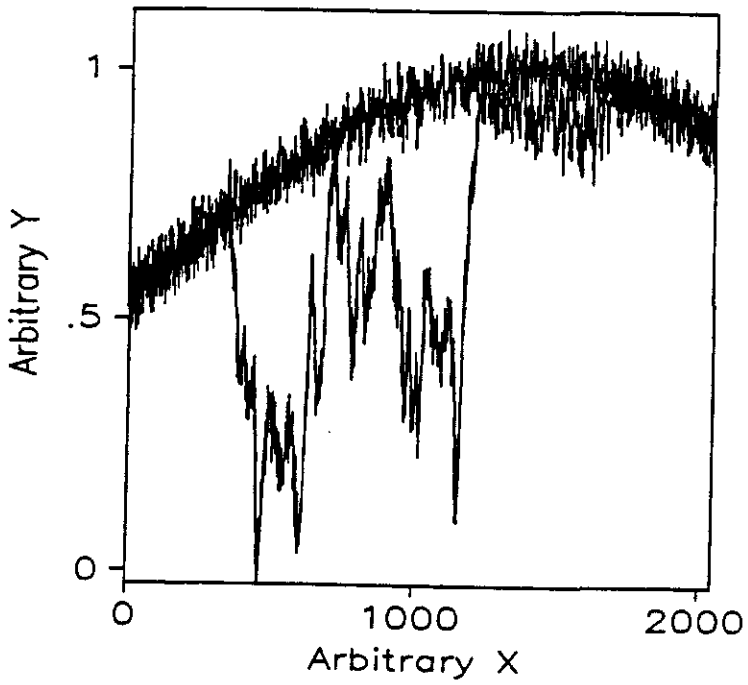
Unnormalized sample spectrum file: SPEC5
Reference spectrum file: REF5
Peakshape file: PEAK5

Using HANDLE:

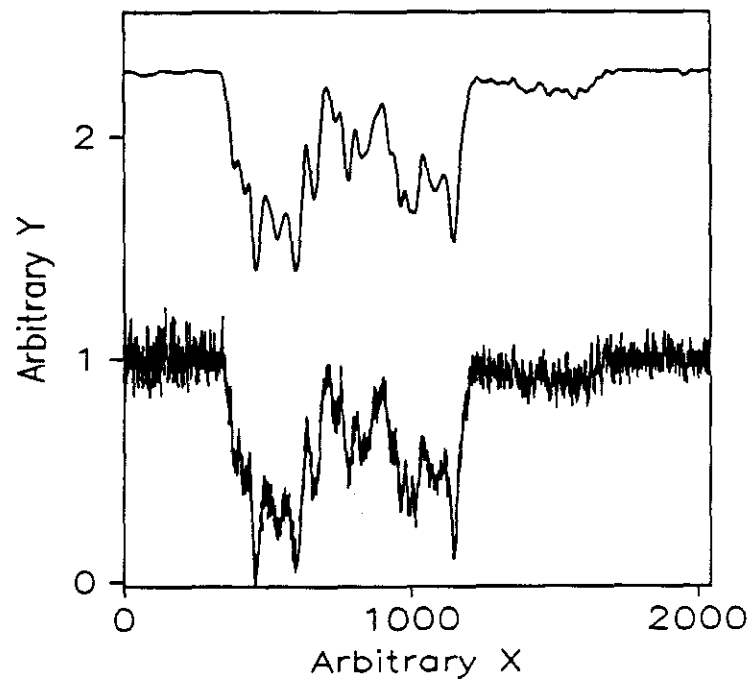
```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
ESM=EntropySMooth. Smooths Normal (thermal/gaussian) noise. ME
PSM=PoissonSMooth. Smooths Poisson (counting) noise. ML.
NSM=NormalSMooth. Smooths Normal noise. 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
     ........
     ........
```

```
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,

$$\mathbf{Sample + noise}$$

$$\mathbf{Reference + noise}$$

Their ratio gives a transmission spectrum,

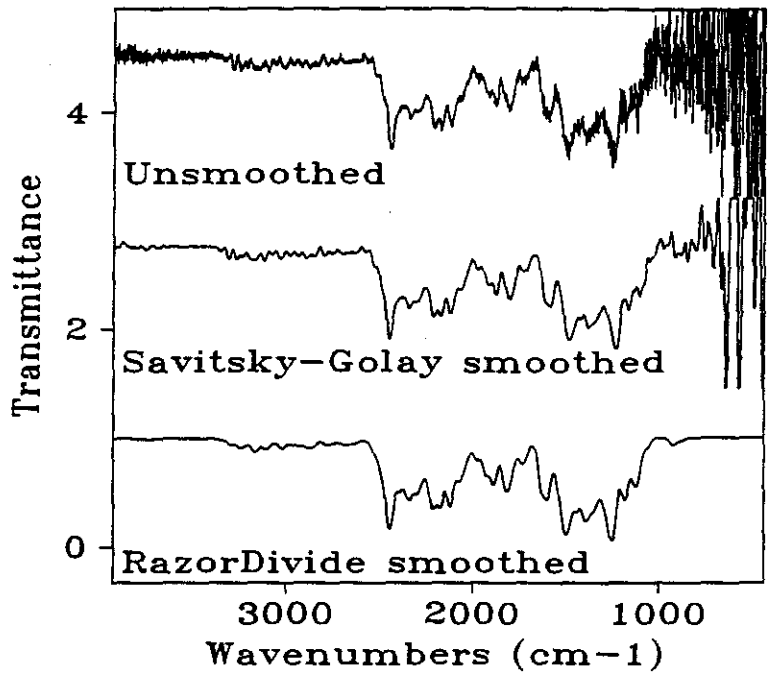
$$\mathbf{T}_j = \mathbf{Sample + noise / Reference + noise},$$

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 **T**, maximize the probability **P** of obtaining the given transmission spectrum $\mathbf{T}_j = \mathbf{S}_j/\mathbf{R}_j$,

$$\mathrm{P(T_j)} = \frac{1}{1 + \frac{4}{\mathrm{N_t}^2}(\mathrm{T_j} - \mathrm{T})^2},$$

where $\mathbf{S}_j$ is the sample absorption spectrum, and $\mathbf{R}_j$ is the reference spectrum. **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 **S** and **R**. Thus, $\mathbf{N}_t = \mathbf{N}_t(\mathbf{N}_s,\mathbf{N}_r,\mathbf{S},\mathbf{R})$. Note that **S**, **R**, $\mathbf{T}_j$, **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 **T** is constrained by

$$\mathbf{T} = \exp(-\mathrm{shape_{ik}} \star \mathrm{o}(\nu_\mathrm{k})).$$

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

$$\mathrm{P(T_j(x_1, x_2, ..., x_n))} =$$

$$\prod_{\mathrm{i=1}}^{\mathrm{n}} \frac{1}{1 + \frac{4}{\mathrm{N_t}^2}(\mathrm{T_j(x_i)} - \exp(-\mathrm{b_{ik}} \star \mathrm{o(x_k)}))^2}.$$

Maximum Likelihood asks and answers this question:

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

The Maximum Likelihood solution is then an estimate of the transmission spectrum **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 ≤ 1000 cm^{-1} nor ≥ 3400 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 aphysical 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, 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 = ((**ydata** - **yout**-convolved-with-**shape**)/**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 (**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 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***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 (**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 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 **rzsizn** will calculate **n**, the minimum amount of space needed. The required size of **n** is determined by **n2** and by the width of the peak in the **shape** array. Obtain the minimum required **n** with this call:

n = rzsizn(n2,shape,nl2)

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

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

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

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

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

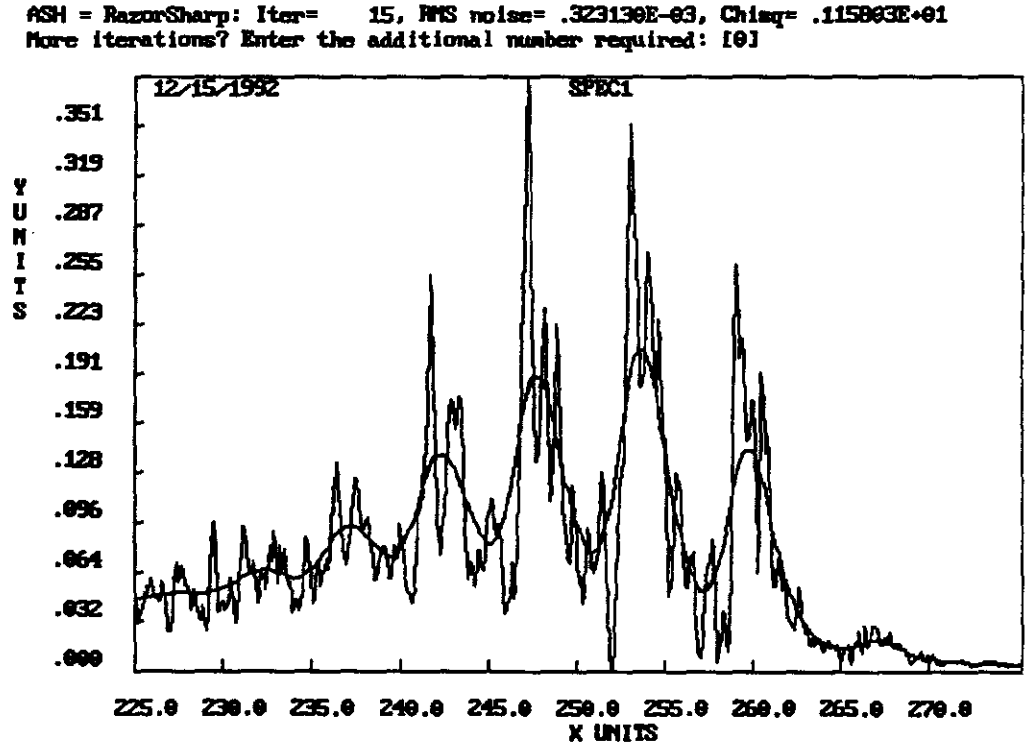
We passed these files through the HANDLE program contained on your disk, using RazorASharp (Command ASH). The result is shown.

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



4.4 rzrdec — RazorDEConvolve

rzrdec 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.

rzrdec 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 **rzrdec** 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 **rzrdec** than for any other function of Razor Library, because the underlying algorithm of **rzrdec** 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.

  rzrdec 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 **rzrdec**.
- Call **rzrdec** with **iter** = 0 for the first iteration only. The function will then maintain **iter** for you. Most data files require 100 to 200 iterations.
- **rzrdec** 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, **rzrdec** 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, 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 = ((**ydata** - **yout**-convolved-with-**shape**)/**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 **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⋆**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 (**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**.

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

n = rzsizn(n2,shape,nl2)

On *output*, **n** is the amount of *space used for the Fourier transforms* in the **yout**, **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 with to process additional data with the same peakshape, you need not change either **newpk** or **n**, provided that (a) your peakshapes do not change, and (b) your input **ydata** sizes (**n2**+1) do not increase.

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

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

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.

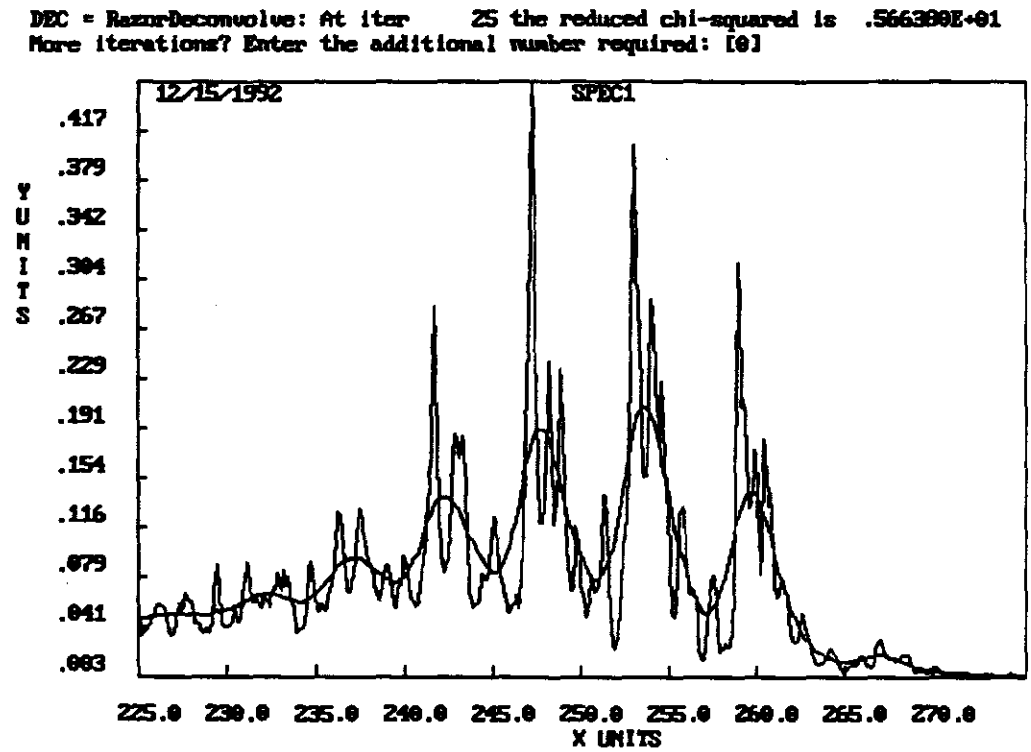
We passed these files through the HANDLE program contained on your disk, using RazorDeconvolve (Command DEC). The result is shown.

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
```

```
        .........
        .........
RZRDEC Iter= 97, RMS= .000171, Chisq = 15.96, Entropy=-.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
```



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 *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, 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 = ((**ydata** - **yout**-convolved-with-**shape**)/**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 (**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 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⋆**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 (**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 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 **rzsizn** will calculate **n**, the minimum amount of space needed. The required size of **n** is determined by **n2** and by the width of the peak in the **shape** array. Obtain the minimum required **n** with this call:

n = rzsizn(n2,shape,nl2)

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

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

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

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

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

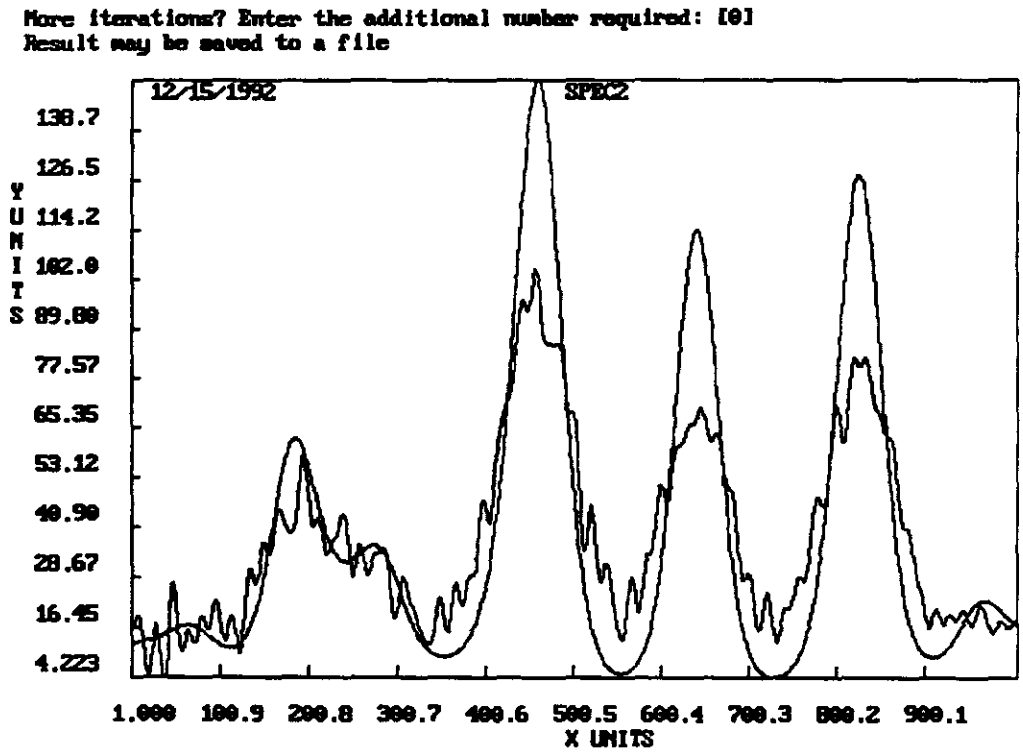
4.7 Example using rzrluc

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
```


More iterations? Enter the additional number required: [0]
Result may be saved to a file
12/15/1992
SPEC2
138.7
126.5
114.2
102.0
89.80
77.57
65.35
53.12
40.90
28.67
16.45
4.223
Y UNITS
1.000 100.9 200.8 300.7 400.6 500.5 600.4 700.3 800.2 900.1
X UNITS

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:

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

$$\mathrm{y_i} = (\mathrm{o} \otimes \mathrm{s})_\mathrm{i} + \mathrm{n_i}.$$

The set $\{o_1, o_2, ...o_n\}$ 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 $\{o_1, o_2, ...o_n\}$.

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

$$\mathrm{p(y_i \mid o)} = \frac{1}{\sqrt{(2\pi)}\sigma_\mathrm{i}} \mathrm{exp}[-\frac{(\mathrm{y_i} - (\mathrm{o} \otimes \mathrm{s})_\mathrm{i})^2}{2\sigma_\mathrm{i}^2}].$$

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

$$\mathrm{p(y_i \mid o)} = \frac{(\mathrm{o} \otimes \mathrm{s})_\mathrm{i}^{\mathrm{y_i}} \mathrm{e}^{-(\mathrm{o} \otimes \mathrm{s})_\mathrm{i}}}{\mathrm{y_i!}}.$$

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

$$\mathrm{p(y_1, ...y_n \mid o_1, ..., o_n)} = \prod_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{p(y_i \mid o)}$$

For Normal noise, this becomes

$$\mathrm{p(y_1, ...y_n \mid o_1, ..., o_n)} = \prod_{\mathrm{i=1}}^{\mathrm{n}} \frac{1}{\sqrt{(2\pi)}\sigma_\mathrm{i}} \mathrm{exp}[-\frac{(\mathrm{y_i} - (\mathrm{o} \otimes \mathrm{s})_\mathrm{i})^2}{2\sigma_\mathrm{i}^2}].$$

For Poisson noise,

$$\mathrm{p(y_1, ...y_n \mid o_1, ..., o_n)} = \prod_{\mathrm{i=1}}^{\mathrm{n}} \frac{(\mathrm{o} \otimes \mathrm{s})_\mathrm{i}^{\mathrm{y_i}} \mathrm{e}^{-(\mathrm{o} \otimes \mathrm{s})_\mathrm{i}}}{\mathrm{y_i!}}.$$

4.11.1 Maximum Likelihood Restoration

When we maximize the probability

$$\mathrm{p(y_1, ..., y_n \mid o_1, ..., o_n)},$$

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

4.11.2 Bayesian and Maximum Entropy Restoration

When we maximize the probability

$$\mathrm{p}(\mathrm{o}_1, ..., \mathrm{o}_n \mid \mathrm{y}_1, ..., \mathrm{y}_n),$$

by invoking Bayes Rule, then we are using a **Bayesian Spectral Restoration** method (also called the MAP (Maximum A Posteriori) method).

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

$$\mathrm{p}(\mathrm{o}_1, ..., \mathrm{o}_n \mid \mathrm{y}_1, ..., \mathrm{y}_n) = \frac{\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_n \mid \mathrm{o}_1, ..., \mathrm{o}_n)\mathrm{p}(\mathrm{o}_1, ...\mathrm{o}_n)}{\mathrm{p}(\mathrm{y}_1, ..., \mathrm{y}_n)}.$$

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

For this application, we will use the multinomial probability law, combined with a prior spectrum $\{Q_1, Q_2, ...Q_n\}$, and so the a priori probability for the parent spectrum is

$$\mathrm{p}(\mathrm{o}_1, ..., \mathrm{o}_n) = \frac{\mathrm{N}!}{\mathrm{o}_1!\mathrm{o}_2!...\mathrm{o}_n!}\mathrm{Q}_1{}^{\mathrm{o}_1}\mathrm{Q}_2{}^{\mathrm{o}_2}...\mathrm{Q}_n{}^{\mathrm{o}_n}$$

.

When we maximize the probability $\mathrm{p}(\mathrm{o}_1, ..., \mathrm{o}_n \mid \mathrm{y}_1, ..., \mathrm{y}_n)$, and use the multinomial probability law for $\mathrm{p}(\mathrm{o}_1, ..., \mathrm{o}_n)$, then we are performing **Maximum Entropy Spectral Restoration.**

In summary, the Maximum Likelihood prescription says we must maximize $\mathrm{p}(\mathrm{y} \mid \mathrm{o})$. The Bayesian prescription says we must maximize $\mathrm{p}(\mathrm{o} \mid \mathrm{y})$. The Maximum Entropy prescription says we must maximize $\mathrm{p}(\mathrm{o} \mid \mathrm{y})$, and additionally use a multinomial probability law as the degeneracy factor for the prior spectrum. The maximizations are to be done under a set of constraints. An important constraint is our knowledge of the peakshapes. We assume that the object function $\{o_1, o_2, ...o_n\}$ is composed of many individual peaks. However, we make no assumption about how many peaks there are, or where they are. We assume that the object function has been convolved with a peakshape function, thereby producing the spectrum we observe. The convolving peakshape function is known to us; its shape is represented by s. We also assume that the object function is positive everywhere.

4.11.3 Razor Library's two restoration methods

Razor Library contains two separate restoration methods, Maximum Entropy/Bayesian, and Maximum Likelihood restoration. The two methods, and the equations which they solve, are given here.

RazorDeconvolve — rzrdec is a **Maximum Entropy/Bayesian** method. It maximizes the probability

$$\mathbf{p(o \mid y)} = \{\prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_i}\mathbf{exp}[-\frac{(y_i - (o \otimes s)_i)^2}{2\sigma_i^2}]\}\{\frac{(o_1 + ... + o_n = N)!}{o_1!o_2!...o_n!}Q_1{}^{o_1}Q_2{}^{o_2}...Q_n{}^{o_n}\},$$

under the constraint that o is positive. $\{Q_1, Q_2, ...Q_n\}$ is the prior spectrum. The Maximum Entropy character becomes evident if we take the logarithm of **p**,

$$\mathbf{ln(p(o \mid y))} = \sum_{i=1}^{n}(-\frac{(y_i - (o \otimes s)_i)^2}{2\sigma_i^2} - o_i\mathbf{ln}o_i + o_i\mathbf{ln}Q_i) + \mathbf{constant\ terms}$$

.

The term

$$-\sum_{i=1}^{n} o_i\mathbf{ln}o_i$$

is the same expression as the Shannon entropy for the spectrum $\{o_1, o_2, ...o_n\}$. Note that maximizing $\mathbf{ln(p(o \mid y))}$ is the same as maximizing $\mathbf{p(o \mid y)}$, because the probability $\mathbf{p(o \mid y)}$ is always positive.

Another form of this Maximum Entropy equation emerges when we take the derivative of $\mathbf{ln(p(o \mid y))}$ with respect to o_i. The derivative equation is

$$\mathbf{d[ln(p(o \mid y))]/do_i} = (-\frac{(y_i - (o \otimes s)_i) \otimes s}{\sigma_i^2} - \mathbf{ln}o_i + \mathbf{ln}Q_i - 1) = 0.$$

Rearranging terms, exponentiating, renormalizing, and then dropping the subscripts produces the familiar ***classic equation of Maximum Entropy*** **deconvolution**:

$$\mathbf{o} = \mathbf{Q exp}(-\frac{(y - (o \otimes s)) \otimes s}{\sigma^2}).$$

rzrdec solves this classic (nonlinear) Maximum Entropy equation.

RazorASharp and RazorLucy — rzrash and rzrluc are both **Maximum Likelihood** methods. They maximize the probability (Poisson noise case):

$$\mathbf{p(y \mid o)} = \prod_{i=1}^{n} \frac{(o \otimes s)_i{}^{y_i}e^{-(o \otimes s)_i}}{y_i!} = \mathbf{maximum},$$

under the constraint that **o** is positive.

rzrash and **rzrluc** use different mathematical algorithms for solving this equation.

4.11.4 rzrdec solution

rzrdec is a solution to the Maximum Entropy equation for Normal noise statistics. **rzrdec** contains the constraint that the solution is positive, and in addition, *allows one to select a prior* $\{Q_1, Q_2, ...Q_n\}$. The prior is the user's best guess, or his prejudices, of the deconvolved solution. (Note that the prior only marginally influences the final solution.)

The equations solved by rzrdec are not new!!! The equations are described by B. Roy Frieden in "Unified Theory for Estimating Frequency of Occurrence Laws and Optical Objects", Journal Optical Society of America, 73, 927-938, July 1983. (**rzrdec** assumes no degeneracy, and therefore uses the classical limit of the equations in Appendix C of this reference). The Maximum Entropy equations are also described by J. Skilling in "Fundamentals of MaxEnt in Data Analysis", Chapter 2 of *Maximum Entropy in Action*, ed. Brian Buck and Vincent A. Macaulay, Oxford Science.

The **algorithm** used by **rzrdec** to solve this classic Maximum Entropy Restoration problem is **entirely new**. The algorithm, which is proprietary to Spectrum Square Associates, is *many times faster* than the classic conjugate gradient methods used previously.

4.11.5 rzrluc solution

rzrluc is a solution to the above Maximum Likelihood Poisson equation. It contains the additional constraint that the solution is positive. **rzrluc** is an appropriate function to use for emission, absorbance, and counting spectra. The **rzrluc** algorithm is as follows:

$$o_i^{k+1} = o_i^k \left(\frac{y}{o \otimes s}\right) \oplus s.$$

The kernal of this solution may be easily obtained from the Maximum Likelihood equation for Poisson noise using

$$\frac{\delta(lnp(y \mid o))}{\delta o} = 0.$$

This algorithm was first described by W. H. Richardson,"Bayesian Iterative Method of Image Restoration", Journal Optical Society America, 62, 55-59, 1972, and shown to converge to the Likelihood maximum by L. B. Lucy, "An iterative technique for the rectification of observed distributions", Astron. Journal 79, 745-765, 1974. It has been named the EM algorithm by L. A. Shepp and Y. Vardi, "Maximum Likelihood Reconstruction in Positron Emission Tomography", IEEE Transactions on Medical Imaging 1, 113-122, 1982. (Astronomers have called it the Lucy algorithm for many years, and we continue to do so here.)

The Lucy algorithm makes good progress for about 10-15 iterations, and then gets bogged down. Although mathematically proven to *eventually* converge, you may grow old waiting!

4.11.6 rzrash solution

rzrash is useful for resolution enhancement of any emission, absorbance, and counting spectra, provided that the convolving peakshape function s is well known.

The **rzrash** algorithm is a solution to the Poisson equation

$$p(y \mid o) = \prod_{i=1}^{n} \frac{(o \otimes s)_i^{y_i} e^{-(o \otimes s)_i}}{y_i!} = \text{maximum}.$$

This is the same equation used by **rzrluc**. However, the **rzrash** and **rzrluc** algorithms are different. You will find that **rzrash** converges to a solution faster than **rzrluc** does.

Technically, **rzrash** is an appropriate function to use for resolution enhancement of data with Poisson noise. However, we have found that it may be used with confidence on Normal noise data also. How do we know this?

We have used a modification of the **rzrash** algorithm to solve the Normal equation

$$p(y \mid o) = \prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_i} \exp[-\frac{(y_i - (o \otimes s)_i)^2}{2\sigma_i^2}] = \text{maximum}.$$

We have compared the Poisson-statistics algorithm **rzrash**, and the corresponding Normal-statistics algorithm on every spectrum we could find, and many that we conjured up ourselves, containing noise of both types. In every case, the differences between the results were small, smaller than the uncertainties associated with our approximate solutions.

We believe we would mislead our users if we presented them with a choice of algorithms — suggesting that the differences in the results were meaningful. Instead, we provide you with you this information about the performance of **rzrash**.

- In most cases, **rzrash** is only capable of increasing resolution by a factor of 2 to 3. The best resolution enhancement which can be achieved depends critically upon the peakshapes. In the worst case, when peaks are Gaussian in shape, **rzrash** is incapable of separating peaks whose centers are closer together than 40 % of the fwhm. In the best case, when the peakshapes have sharp features such as seen in the triangular shape of characteristic of the transfer function of a slit spectrometer, resolution enhancement by a factor ≥ 5 is easy.

- Peak areas *may* be in error by as much as 2 % of the largest peak in the data, even in the absence of noise. (Usually they are better than this. The actual performance depends upon peakshapes, and amount of overlap.) When you need better peak areas, we suggest that you use **rzrash** to help you estimate the number of peaks present, and to find the center positions, which it does exceptionally well. Then use RZRFIT to get accurate areas.

rzrash contains our Poisson algorithm. We decided to put this one in for these reasons: (1) The results we obtain, using the Poisson algorithm and the Normal algorithm on the

same data, are indistinguishable in most cases. (2) A Poisson error curve is nearly the same as a Normal error curve when the signal is large. (Clearly, this is one reason our two algorithms have nearly the same performance.) (3) The hardest cases are those with Poisson statistics and low signal/noise. If anyone is down there pushing the limit, he may need the Poisson algorithm. (4) Our Poisson algorithm is more stable for highly asymmetric peakshapes. (5) The Poisson algorithm converges more quickly than the Normal one.

4.12 What about Fourier deconvolution?

Fourier deconvolution and Maximum Likelihood Restoration begin with the same problem. There is an observed data set $\{y_1, y_2, ...y_n\}$, where

$$\mathrm{y_i} = (\mathrm{o} \otimes \mathrm{s})_\mathrm{i} + \mathrm{n_i},$$

where $\{o_1, o_2, ...o_n\}$ is a more highly resolved spectrum, $\otimes$ denotes convolution, s describes the shapes of the peaks, and n_i are the noise fluctuations.

Fourier deconvolution estimates the more highly resolved spectrum in this way: Ignore the noise, and solve the equation

$$\mathrm{y_i} = (\mathrm{o} \otimes \mathrm{s})_\mathrm{i}.$$

The solution is obtained in the Fourier domain, by dividing the Fourier transform of the data $\{y_1, y_2, ...y_n\}$ by the transform of the peakshape $\{s_1, s_2, ...s_n\}$. The inverse transform of the quotient is the estimate of $\{o_1, o_2, ...o_n\}$. Problems arise. The noise, which has been ignored, doesn't go away. It dominates the signal at large Fourier frequencies, and is now amplified. The inverse transform produces ringing, and negative intensities, in $\{o_1, o_2, ...o_n\}$. These problems are 'solved' by applying a filter in the Fourier domain before doing the inverse transform. The filter function is arbitrary. The art of Fourier deconvolution is tinkering with the filter.

Maximum Likelihood Restoration and Fourier deconvolution are two different methods of solving exactly the same problem. They both begin with the assumption that the observed spectrum is the sum of many peaks which have a known shape s. There is no underlying pedestal or baseline. Note that the equation

$$\mathrm{y_i} = (\mathrm{o} \otimes \mathrm{s})_\mathrm{i}$$

doesn't intrinsically require that all peaks have the same shape. The requirement that all peaks have the same shape, and width, is added so that one can use fast Fourier transforms (FFTs) for the convolution operation. Fourier deconvolution wouldn't work without this assumption. And you wouldn't want to wait long enough for **rzrash** to perform convolutions without the aid of an FFT.

Maximum Likelihood Restoration and Fourier deconvolution differ in these ways: Fourier deconvolution ignores the noise, and then applies an arbitrary filter in the Fourier domain to clean things up. Maximum Likelihood Restoration acknowledges the noise, looks for the solution which has the highest probability of being correct, and imposes the additional condition that $\{o_1, o_2, ...o_n\}$ is positive (the source didn't put out negative intensities).

4.13 A Final Word of Advice

The secret of success in spectral restoration is: don't guess. Guessing involves tinkering with the process until you like the result, get tired, or no longer believe anything you see. Everyone who goes this route eventually ends up in the last state, and properly so. Decide upon your constraints, run the program, and if the results still fall short of your needs then either you need better data, or a different algorithm better adapted to your particular constraints, or you have a research-grade problem.

4.14 rzrstr — RazorStrip

RazorStrip (**rzrstr**) is a linear deconvolution method, similar to Fourier deconvolution. The main differences between **rzrstr** and Fourier deconvolution are (a) **rzrstr** is less sensitive to noise, due to its Maximum Entropy roots, and (b) **rzrstr** does not have any arbitrary, user-selected parameters.

rzrstr is obtained by expanding the exponent in the ***classic equation for Maximum Entropy* deconvolution** (p. 68), and then using the first two terms of the expansion. We don't think that **rzrstr** is an optimum method of deconvolution[1]; **rzrdec** is better. However, we provide **rzrstr** as an excellent alternative for users who might need a linear method, and for users who are still tempted by Fourier deconvolution (or FSD).

We believe that if you are bound and determined to use a linear deconvolution method, then **rzrstr** is the best choice. Best of all, **rzrstr** has no tinkering parameters!!

Required user input:

- Data set in which any **baseline or offset has been properly removed.** Remember that **rzrstr** *and* Fourier deconvolution are based upon the assumption that the data file contains a set of overlapping peaks of the same shape, but no baseline!
- 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.

[1]Thus we have hidden **rzrstr** here in the back of the deconvolution chapter. We hope you will turn to **rzrdec** instead.

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

Input arrays which must be filled:
ydata, filled between **0** and **n2**
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 **yout** 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, 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
sigma = RMS noise in **ydata**.
Function return values:
rzrstr = 0 if iteration was successful
If **rzrstr** < 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 NOT be altered.

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*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 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 thus **trans** is newly loaded by **rzrstr**. When **newpk** > 1, it is expected that **trans** has not been changed since the last time it was filled. See the discussion below for **newpk**.

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

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

n = rzsizn(n2,shape,nl2)

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

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

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

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

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1*. Whenever **rzrstr** 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 **rzrstr** will be **rzrstr** = -2.

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

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**.

Chapter 5

RazorDerivative — rzrdif

5.1 A Fundamental Approach to Derivatives

There seem to be many, arbitrary ways to create a derivative from an array of numbers, ranging from a simple two-point difference to a Savitsky- Golay derivative with m-point polynomial-smoothing. Razor Library employs a different approach, one which uses only the fundamental knowledge available to the spectroscopist. The **rzrdif** derivatives answer the following question:

If I assume that my data set **ydata** consists of peaks like the one in **shape**, *what does the most 2nd likely derivative (or 3rd derivative, etc...) look like?*

The differences between this statistical method, and other methods, are:

- The statistical approach doesn't force the user into an arbitrary choice of methods.
- The statistical approach has no arbitrary parameters.

The mathematics which implement statistical derivatives are explained in Section 5.3, on page 84.

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

- Data array.
- Peakshapes - either true or estimated. It is not critical that the user choose an exact peakshape for **rzrdif**. 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.

Programming notes:

- Set **nord** = 2 to calculate the 2nd derivative, etc.
- The programmer will need to provide two full-size processing arrays, **yout**, and **trans**, as well as the data array **ydata**. **ydata** will *not* be altered by **rzrdif**.
- You may SAVE SPACE by allowing the derivative, which is returned in the array **yout**, to replace the input **ydata**. Pass the **ydata** address again, instead of an address to a separate array **yout**, in the fifth argument position. Ensure that the size of **ydata** array is $\geq$ **n**. In this case, **ydata** will be altered outside the range **(0,n2)**.

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

Input arrays which must be filled:
ydata, filled between **0** and **n2**, length **n**
shape, filled between **nl1** 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: **n1, n2, n, nl2, npks, psens, newpk**
n2 is the last position of data in **ydata**
nl2 is the last position of data in **shape**
n is the size of arrays **yout**, and **trans**
newpk indicates whether or not **shape** is a new peakshape.
nord is the ordinal number of the desired derivative.
Output arrays:
yout, filled with the desired derivative
Output variables:
n = amount of array space used
NOTE: if **n** is negative, abs(n) = amount of array space needed (but not available). Operation not successful.
newpk = n if **trans** was loaded successfully.
nfwhm = full-width-at-half-maximum of peakshape in **shape**
sigma = RMS noise in the **ydata**.
Function return values:
rzrdif = 0 if operation was successful
If **rzrdif** < 0, error occurred
Use **rzrerr** (page 174) to obtain error text

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.

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**.

yout is the *output derivative.* The derivative will be found between data points **0** and **n2**, and should be ignored outside this range.

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

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

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

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

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

n = rzsizn(n2,shape,nl2)

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

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

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

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

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1*. Whenever **rzrdif** 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 **rzrdif** will be **rzrdif** = -2.

nord is *input* as the ordinal number of *the desired derivative*. For example, set **nord** = 2 to obtain the 2nd derivative.

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

sigma is *output* as the *standard deviation (root- mean-square) of the noise* which was found in **ydata**.

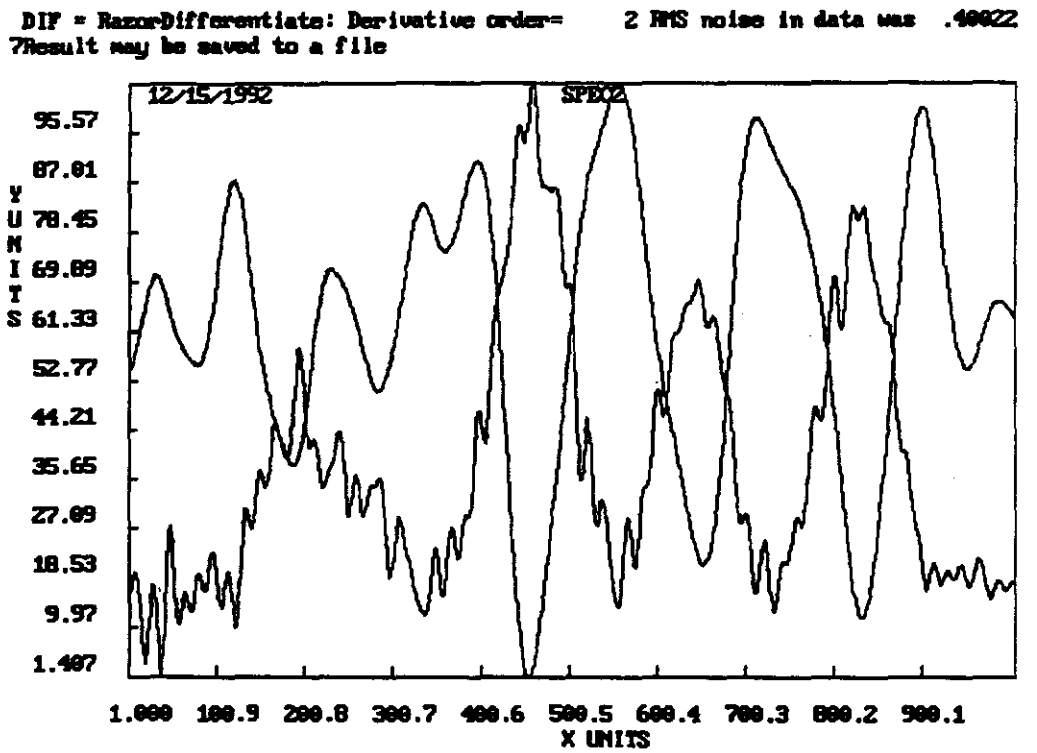
5.2 Example using rzrdif

HANDLE is set up to use **rzrpic** to pickk peaks from a spectrum, ask you a few questions, and then pass the chosen peaks into **rzrfit**. **rzrpic** uses a form of the Maximum Likelihood/Bayesian 2nd derivative calculated by **rzrdif**. You may obtain your own 2nd derivative to use in peak selection. The example here uses the file SPEC2, which is an ICP spectrum of iron.

Spectrum 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. 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): DIF
Enter name of unnormalized sample spectrum (Try SPEC2): SPEC2
Enter name of peakshape file (Try SPEC2): PEAK2
Enter derivative order 1=1st, 2=2nd, ... : 2
Entering RZRDIF. Wait for processing...
DIF = RazorDerivative
The estimated RMS noise is 3.4088
The FWHM of the peakshape is 80
The computed derivative = 2 (1st,2nd,etc)
The size of array space used was 2048
RESULT MAY BE SAVED TO A FILE
```


DIF = RazorDifferentiate: Derivative order= 2 RMS noise in data was .40022
?Result may be saved to a file
12/15/1992
SPEC2
95.57
87.01
78.45
69.89
61.33
52.77
44.21
35.65
27.09
18.53
9.97
1.407
UNITS
1.000 100.9 200.8 300.7 400.6 500.5 600.4 700.3 800.2 900.1
X UNITS

5.3 Equations of Bayesian Derivatives

(This is a summary of a paper given at the 1991 Pittsburgh Conference.)

We will show how we find the Bayesian second derivative of a given data set. Other derivatives are found in a similar manner.

We begin by setting up an equation which describes the probability of obtaining the data set $d(x_1), d(x_2), ...d(x_n)$, in terms of the parent spectrum $y(x)$.

From a parent spectrum $y(x)$, we have drawn a data set $d(x)$ containing data points $d(x_1), d(x_2), ...d(x_n)$, where $d(x_i) = y(x_i) + n_i$. The $d(x_i)$ are the data values, and n_i are the noise values.

We assume that the parent distribution $y(x_i)$ consists of a set of peaks of some shape – not necessarily all the same shape - plus a baseline:

$$\mathbf{y(x) = o(x) \otimes s(x) + b(x),}$$

where $o(x)$ = an object function = a set of delta-functions, $\otimes$ means convolution, $s(x)$ = the peakshapes, and $b(x)$ = the baseline.

For a given parent spectrum, the probability $\mathbf{p}$ for the sample $d(x)$ is determined by the probability distributions for the noise $\{n_1, n_2, ...n_n\}$. $\mathbf{p}$ is called the **Likelihood**.

If the noise n_i is random, and additive, with a Normal distribution, then the probability (likelihood) for $d(x_i)$ is

$$\mathbf{p(d(x_i) \mid y(x_i)) = \frac{1}{\sqrt{(2\pi)}\sigma_i} exp[-\frac{(d(x_i) - y(x_i))^2}{2\sigma_i^2}]}.$$

Assume that the noise n_i is uncorrelated with the noise n_j, for all i, j. Then the likelihood of observing the set $\mathbf{d} = \{d_1, d_2, ...d_n\}$ is the product of the probabilities for each of the $d(x_i)$:

$$\mathbf{p(d \mid y) = \prod_{i=1}^{n} p(d(x_i) \mid y(x_i))}$$

For Normal noise, this becomes

$$\mathbf{p(d \mid y) = \prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_i} exp[-\frac{(d(x_i) - y(x_i))^2}{2\sigma_i^2}]},$$

We wish to find the best possible estimate of the second derivative. Thus we will maximize the probability

$$\mathbf{p(y'' \mid d).}$$

We will do this by invoking Bayes Rule. This is the Bayesian method.

Bayes Rule says that the probability $\mathbf{p(y'' \mid d)}$ is related to the probability $\mathbf{p(d \mid y'')}$ through

$$\mathbf{p(y'' \mid d) = \frac{p(d \mid y'')p(y''))}{p(d)}},$$

where

$$\mathbf{p(d \mid y'') = \prod_{i=1}^{n} \frac{1}{\sqrt{(2\pi)}\sigma_i} exp[-\frac{(d(x_i) - \int\int y''(x_i))^2}{2\sigma_i^2}]}.$$

In order to solve the equation, we must also provide the *a priori* probabilities $\mathbf{p(d)}$ for our observed spectrum and $\mathbf{p(y'')}$ for all parent derivative spectra. Clearly, the probability for our single observation is $\mathbf{p(d) = 1}$. What is needed is an *a priori* statement about y''.

This seems to be a reasonable statement about y'': In the absence of data, we don't want to find any peaks in y''. Consequently, in the absence of data, we want

$$\mathbf{y'' = ax + c},$$

which means that

$$\mathbf{y'''' = 0}.$$

Translating this statement into an equation we can use brings us:

$$\mathbf{P_0 = \prod_{i=1}^{n} exp[-\frac{(y''''(x_i))^2}{2\sigma_4^2}]}.$$

The only problem left is to choose σ_4 wisely. To obtain a value for σ_4 , we used the following equations:

$$\mathbf{d(x_i) = y(x_i) + n(x_i)},$$

$$\mathbf{\Sigma_{i=1}^{N} n(x_i)^2 = N\sigma^2},$$

$$\mathbf{\Sigma_{i=1}^{N} n(x_i) = 0}.$$

To get a final solution for the second derivative y'', and to maintain consistency with our assumptions about y'', we also used a baseline that was of the form,

$$\mathbf{b(x) = ax + c}.$$

The final solution to the Bayesian equations shown above gives a transformation T, which transforms the data d into its Bayesian second derivative:

$$\mathrm{y''(x) = T(d(x))}.$$

The transformation **T**is implemented in **rzrdif**, and also used by **rzrpic**. **T**depends upon the noise in the data, and also weakly depends upon the peakshape function $s(x)$. The dependence upon the noise in the data was exactly what we expected. It means that the second derivative will be optimally smoothed. The fact that our final transformation was only weakly dependent upon $s(x)$ was a lucky break. Because the shape of our final second derivative y'' was not overwhelmed by the shape we assumed for s, we were able to use the second derivative to estimate peak widths as well as peak heights. Consequently, we have been able to give you peak width and peak height estimates in **rzrpic**.

Chapter 6

RazorPick — rzrpic

6.1 Accurate Peak-Picking for Merged Peaks

The human eye and brain are the best peak pickers. Previous computer programs designed for this task have set up parameters such as slope, threshold, minimum area, skim/drop decision trees, etc. Even then, the programs turn helpless when given very noisy data. Why can't the computer be more like ourselves?

We have applied Maximum Likelihood and Bayesian methods to this problem. Maximum Likelihood is a mathematical formulation of the same statistical and *a priori* knowledge the brain uses. The human observer discriminates between peaks and noise, and judges when peaks overlap, by the peak shapes. His decision about whether to accept small peaks is based upon probability factors.

RazorPick uses a Bayesian 2nd derivative to find candidate peaks. It then uses the statistics of the noise (Maximum Likelihood) to generate a significance (= signal/noise ratio) for each candidate peak. Finally, it sorts the peaks in order of decreasing significance, and returns a list of all peaks which meet the acceptance criteria.

RazorPick is a collection of 4 excellent peak-picking algorithms. **RazorPick** is superior to most other peak-pickers in the following ways:

- Detects peaks even when merged or overlapping.
- Reports peak significances in signal/noise units, for Normal or Poisson noise.
- Estimates and reports peak heights and peak widths, using information from the (Bayesian) 2nd derivative.
- Identifies positive peaks when presented with a positive template; identifes negative peaks when given a negative template.

6.2 rzrpic

Required user input:

- Select a peakshape which represents the narrowest peaks in the data set. The actual peakshape is not very critical for this algorithm.
- When the selected peakshape is positive, **rzrpic** will search for positive peaks; when the peakshape is negative, negative peaks in the data will be identified.

Processing notes:

- The **High-Performance** picker (**iperf** = 1) finds overlapping peaks if the peak centers are not closer together than about one-half of a peak width. This picker is recommended for data containing peaks of different widths.
- The **High-Resolution** picker (**iperf** = 2) finds overlapping peaks if the peak centers are not closer together than about one-quarter of a peak width. (The actual resolution will depend on the peak shape.)
- In the **2nd-Order High-Performance** picker (**iperf** = 3), an extra asymmetry correction is applied. For symmetric peaks, the High-Performance and 2nd-Order High-Performance modes are exactly the same. For asymmetric peaks, sometimes the extra correction is helpful, and sometimes not. When helpful, it provides extra resolution. (NOTE: both the High-Performance and High-Resolution pickers have 1st-order asymmetry corrections built in!)
- The **Quiet Picker** (**iperf** = 4) is a Bayesian picker tailored for narrow peaks (widths < 5 datapoints).
- The **Quick-Pick** picker (**iperf** = -1) does not separate overlapping peaks, mainly because it is not using a Bayesian 2nd derivative.

Programming notes:

- Set **istat** = 1 if the noise is Normal. Set **istat** = 2 if the noise is Poisson.
- Set **psens** = 3 to find all the peaks with heights > 3 times the RMS noise, i.e. all peaks with signal/noise ratios > 3. Set **psens** = -3 to find all the peaks with areas > 3 times the RMS area-noise.
- The small array **locpks** returns useful information for setting up an automated peak picker. For instance, the 5 most significant peaks will be the first 5 peaks in **locpks**. Use **rzpkst** (Page 175) to resort **locpks** and **sigpks** by location, height, or width.
- Peak significances, heights, and widths are returned in the **sigpks** array. The heights and widths are useful in loading the **datmat** matrix for **rzrfit**. Use **rzdfil** (Page 177) for assistance in loading **datmat**.

long rzrpic(float ydata[ ], long n2, float shape[ ], long nl2,
float yout[ ], float w[ ], float trans[ ],long *n, long *newpk,
long *istat, long iperf, double *psens, long locpks[ ], long *npks,
float sigpks[ ], long nsig, long *nfwhm, double *peak, double *sigma)

Input arrays which must be filled:
ydata, filled between **0** and **n2**, length **n2** + **1**
NOTE: **ydata** will be read only, not altered.
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**
locpks, length **npks**
sigpks, length **nsig**
Input variables: **n2, nl2, n, newpk, istat, iperf, psens, npks, nsig**
n2 is the last position of data in **ydata**
nl2 is the last position of data in **shape**
n is the size of arrays **yout, w**, and **trans**
newpk indicates whether or not **shape** is a new peakshape.
istat is a flag for Normal vs. Poisson noise.
iperf is a flag for High-Performance/High-Res/2nd-Order/Quiet/Quick-Pick.
psens is the threshold peak sensitivity in S/N units.
npks is the size of the **locpks** array.
nsig is the size of the **sigpks** array.
Output arrays:
yout, filled with a linear baseline
w, filled with smoothed data between **0** and **n2**
locpks, filled with locations of identified peaks
sigpks, filled with peak significances, heights, and widths
Output variables:
n = amount of array space used
NOTE: if **n** is negative, abs(**n**) = amount of array space needed (but not available). Operation not successful.
newpk = n if **trans** was loaded successfully.
npks = number of peaks detected
nfwhm = full-width-at-half-maximum of peakshape in **shape**
peak = height of peakshape in **shape**
sigma = RMS noise in the **ydata**.
Function return values:

rzrpic = 0 if operation was successful
If **rzrpic** < 0, error occurred; use **rzrerr** (page 174) for 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.

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**. **shape** may be a subarray within **ydata**. If the peakshape is right-side up, positive peaks will be identified by **rzrpic**. If the peakshape is a negative peak, then negative peaks will be found.

nl2 is *input*. It is the *index of the last data point of the peakshape* in **shape**.

yout is an *output* array, filled between **0** and **n2**. **yout** will contain the linear baseline used by the picker.

yout must have minimum size **n**. See the discussion below for **n**.

w is a *work array* with minimum length **n**.

On *output*, **w** contains a *smoothed data file*. The smoothing has been done by **rzrpic**.

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**.

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

n = rzsizn(n2,shape,nl2)

On *output*, **n** is the amount of *space used for the Fourier transforms* in the **yout**, **w**, and **trans** arrays. If **n** is negative on output, the amount of space furnished

was inadequate, and no processing has taken place. If **n** is returned negative, then abs(**n**) is the amount of space needed in the above arrays.

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

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

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

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1*. Whenever **rzrpic** 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 **rzrpic** will be **rzrpic** = -2.

istat is an *input flag which governs the statistics* used by the function. Set **istat** = 1 if the noise is Normal. Set **istat** = 2 if the noise is Poisson.

iperf is an *input flag which governs the performance mode* of the function. Set **iperf** = 1 to get the High-Performance Bayesian picker. **iperf** = 2 gives a High-Resolution Bayesian picker. **iperf** = 3 brings up the 2nd-Order High-Performance Bayesian picker. **iperf** = 3 is the Quiet Picker. Set **iperf** = -1 to get a Quick Pick. See the discussion under 'Processing notes' on page 88.

psens is an *input S/N threshold variable* that directs the peak picker. The peak picker assigns each peak a significance in units of the RMS noise. **rzrpic** returns peaks whose significances exceed the value **psens**. When **psens**=0.0, all possible peaks are found. When **psens** = 3.0, all peaks with heights > 3.0 RMS noise (i.e. S/N > 3.0) are returned. When **psens** = -3.0, all peaks with areas > 3.0 RMS area-noise are returned. Peaks which are at least 3 to 5 times the RMS noise are meaningful (**psens** = 3 to 5).

locpks is an *output integer array containing the peak locations*, i.e., **locpks**[0] = data point number of the first peak detected. **locpks** need be no larger than the maximum number of peaks expected. **locpks** and **npks** may be used as input to **rzrfit**.

npks is *input* as *the size of array* **locpks**.

npks is *output* as the *number of peaks* located by the search. Thus, the array **locpks** will be filled with meaningful numbers between **locpks**[0] and **locpks**[**npks**-1].

sigpks is an *output array containing the peak significance* assigned by **rzrpic**. The significance is in units of the RMS noise in the data set. The output arrays **locpks** and **sigpks** are sorted by significance. **sigpks**[0] $\geq$ **sigpks**[1], etc.

The length of the **sigpks** array should be 3***npks** = three times the maximum number of peaks expected. This will provide room to report the peak significances, peak heights and peak widths.

The contents of the **sigpks** array will be **sigpks**[0] = significance assigned to the peak found at position **locpks**[0], etc..., **sigpks**[**npks**] = height assigned to the peak found at position **locpks**[0], etc..., **sigpks**[**npks***2] = width assigned to the peak found at position **locpks**[0], etc.

nsig is *input* as *the size of array* **locpks**.

The minimum length of the **sigpks** array is **nsig** = **npks**. This provides enough room to return peaks significances in **sigpks**.

To obtain peak heights and peak widths in **sigpks**, as well as peak significances, set **nsig** = 3***npks** = three times the maximum number of peaks expected.

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

peak is *output* as the *height of the peakshape* in the array **shape**.

sigma is *output* as the *standard deviation (root-mean-square) of the noise* which was found in **ydata**.

6.3 Example using rzrpic

HANDLE is set up to use **rzrpic** to pick peaks from a spectrum, ask you a few questions, and then pass the chosen peaks into **rzrfit**. We used the file SPEC2, which is an ICP spectrum of iron. The human eye says that the left peak is double, and the peak picker easily found it. We suggest that the programmer display the peaks on the screen, and ask the user to edit the choices.

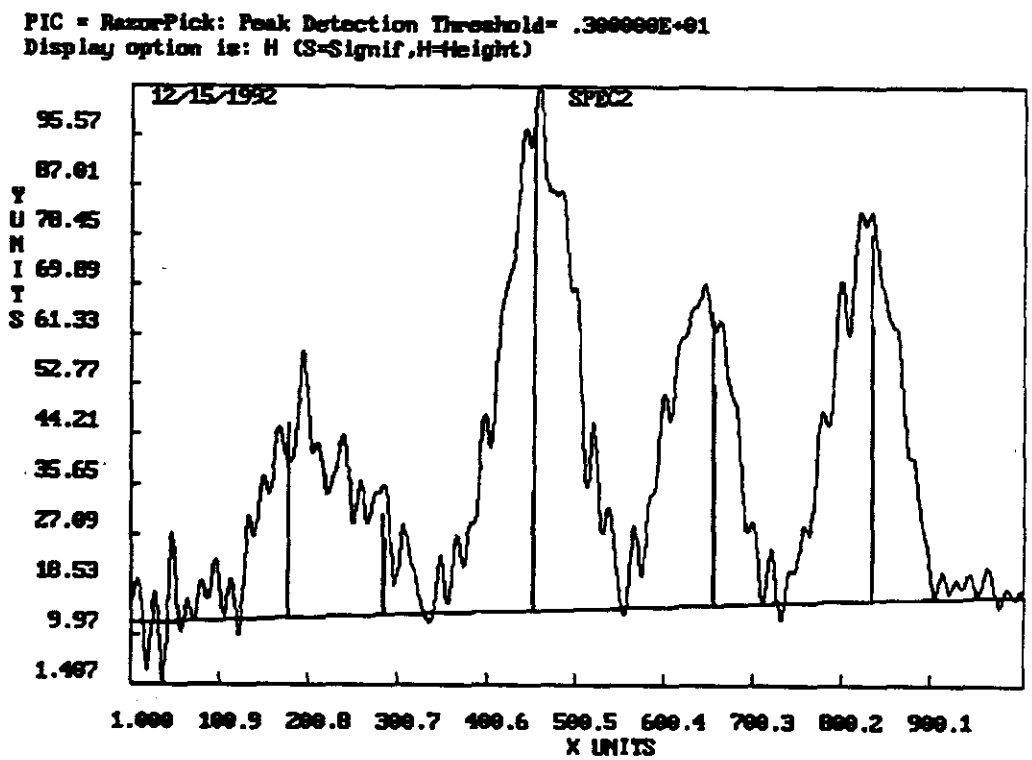
HANDLE doesn't have a graphical interface, so you will have to use the picture below to select peaks from its choices. Using HANDLE, you can't select exactly the peaks you want, only the first nn from a list which is presented. We accepted all five. The results were then passed to **rzrfit**, described in the next chapter.

Spectrum 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): PIC

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

Enter -1 for Quick-Pick
Enter 1 for High-Performance picker
Enter 2 for High-Resolution picker
Enter 3 for 2nd-Order High-Performance picker
```

```
Enter  4  for Quiet-Pick (High Res. Good for very narrow peaks.)
Select picker [2]: 2

Enter peak threshold (# of standard deviations of noise)
Enter 3 if unsure: 3

Enter N for Normal noise; P for Poisson noise [N]: N

Entering RZRPIC.  Please wait for processing...
Estimated RMS noise: 3.4088
# Peaks detected: 5
Using Peak detection threshold: 3.0
Peak HEIGHT Significances:
    24.22    18.41    14.58     9.62     5.04
Number of peaks(as sorted) accepted for FIT: 5
Select:
Print (S)ignificances (L)ocations (H)eights, (W)idths, (E)verything
(R)esort. (T)uneup heights. (A)ccept. (M)enu.
```



Chapter 7

RazorFit — rzrfit

7.1 Accurate Peak Areas, with Confidence Limits

RazorFit is a peak-fitting algorithm. It fits a model consisting of a sum of peaks of various shapes, and a baseline, to the given data set. This is the preferred method when accurate areas are required. **RazorFit** is a Maximum Likelihood technique for Normal noise statistics.

RazorFit has abilities not found in other peak fitting functions:

- Lets you fit *real data* peaks. You may capture a data peak from any file, or from your current data, and then use the shape of your Captured DataPeak as a template for fitting peaks in your data. When you fit your real data peaks with shapes that really match, you obtain more accurate areas. **Only RazorFit can do this!**

- Processes an arbitrarily large number of peaks in large data arrays, by automatically arranging the peaks in 'bunches', and processing the bunches sequentially. Bunch processing is much faster than all-at-once processing, It is a successful tactic for situations where peak-fitting algorithms often fail: namely, on data files with large numbers of peaks. **Only RazorFit can do this!**

- Allows 'linking' peaks together in a master/slave relationship. For example, the user may specify that slave-peak-2 is always found a constant distance away from master-peak-1, or that slave-peak-8a is always 1/2 the height of master-peak-7, while slave-peak-8b is always twice the height and 14 times the width of master-peak-7. The master/slave relationship may be established for only one parameter, or for any combination of parameters, of any two peaks. 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. **Only RazorFit can do this!**

- Understands Poisson (counting) statistics, and will correctly minimize the chisq statistic for Poisson noise, even if the user has *rescaled* the data. **Only RazorFit can do this!**
- Admits a wide variety of peak shapes, including asymmetric shapes, and user-supplied peakshapes. Note that use of the proper peak types, and proper baselines, is the key to obtaining accurate areas for *all* fitting algorithms.
- Allows any number of parameters (i.e., peak positions) to remain fixed, as is appropriate if their values are known from other considerations.
- Provides accurate areas, positions and widths, with corresponding confidence limits, of each component peak.

7.2 rzrfit

rzrfit fits parameterized model peaks to a spectrum. The model function(s), and number of peaks, must be specified by the user. Initial parameter estimates must be furnished; these are refined by iteration using the Levenberg-Marquardt algorithm.

A maximum of six free parameters for each model peak are permitted by **rzrfit**. If you have an analytical expression for a model peak shape that you wish to use for fitting your data, you may write your own peakshape function. See **rzrser02.c** for instructions for the writing of, and examples of, model peakshape functions.

Use **RazorPick** to give a starting estimate of the number and positions of all peaks, including hidden components. This automatic peak-picker frees the user or programmer from the tedious task of selecting starting parameters for the peaks which form the model.

long rzrfit(float ydata[ ], long n2, long m, float datapeak[ ], long nl2, long ml, float yout[ ], float vnoise[ ], float baslin[ ], float w[ ], long *n, long *ifast, long *istat, float datmat[ ][40], float covar[ ], float hess[ ], long iwork[ ], float work[ ], long mmax)

Input arrays which must be filled:

ydata, filled between **0** and **n2**, length **n2+1**

datapeak, optionally filled between **0** and **nl2**

datapeak length (**nx2+1**) = **n2+1** if filled, *else* = **1**

If running in fast mode (**ifast=1**), **datapeak** MUST be filled.

If using Bunch processing (Page 101), **datapeak** MUST be filled.

If any peak has **type** = 0, **datapeak** MUST be filled.

NOTE: **datapeak** will be read only, not altered.

vnoise, filled with noise variance between **0** and **n2** when **istat = -1**,

If noise is Normal or Poisson, **istat** = 1 or 2, **vnoise** may have length 1.

For Normal noise, **istat = 1**, **vnoise[0]** MUST be filled with either the noise variance, or 0.0. For Poisson noise, **istat = 2**, **vnoise[0]** MUST be filled with either the data scale factor, or 0.0

— see discussions below for **istat** and **vnoise**

baslin, optionally filled between **0** and **n2**,

baslin length = **n2+1** if filled, *else* = **1**

If the baseline has **type** = 200, **baslin** MUST be filled.

datmat, a matrix, filled as discussed on page 100

Additional arrays to be furnished:

yout, length $\geq$ n, if **ifast** = 1 or nbunch > 0. (See p. 101 for nbunch.)

length = n2, if **ifast** = 0 *and* nbunch=0.

w, length $\geq$ n, if **ifast** = 1 or nbunch > 0.

length = 1, if **ifast** = 0 *and* nbunch=0.

covar, length $\geq$ [**mmax*mmax**]

hess, length $\geq$ [**mmax*mmax**]

iwork, length $\geq$ 6***mmax**

work, length $\geq$ 9***mmax**

Input variables which must be filled:

n2, m (=1), nl2, ml (-1), n, ifast, istat, mmax

mmax is the maximum number of parameters needed to calculate the model.

mmax = 6***npks** is a safe choice for All-at-once processing.

mmax = 6***nbunch** is a safe choice for Bunch processing.

Bunch processing is discussed on Page 101.

Output arrays:

yout, filled between **0** and **n2**

datmat, filled with peak parameters and standard errors

Function return values:

rzrfit = 0 if there are no errors
If **rzrfit** < 0, error occurred
Use **rzrerr** (page 174) to obtain error text

Description of variables

ydata is the *input* array, which is to be fit between **0** and **n2**.

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

m is the *number of rows* of data to be fit in the **ydata** array (matrix). **m** is furnished as *input*. This parameter will allow for 2-dimensional fits *in the future*. For now, use **m = 1**.

datapeak 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 **datapeak**.

The **datapeak** array is used in the fast mode, when **ifast** = 1, and also used for Captured DataPeaks, when **type** = 0. If there are no peaks with **type** = 0, and if **rzrfit** is being used in the standard mode, **datapeak** will not be used, and then it may be a dummy array of size 1.

ALSO SEE THE DISCUSSION BELOW FOR **ifast**.

ALSO SEE THE DISCUSSION ON PAGE 102 FOR **type**.

ALSO SEE THE DISCUSSION ON PAGE 101 FOR **bunchflag**.

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

nl2 is used in the fast mode, when **ifast** = 1, and also used for Captured DataPeaks, when **type** = 0.

ml is the *number of rows* in the **datapeak** array (matrix). **ml** is furnished as *input*. This parameter will allow for more than one datapeak to be used *in the future*. For now, use **ml = 1**.

yout is the *output* fitted model, the sum of the chosen model peak shapes.

vnoise is an array with length = **n2+1** if **istat** = -1, else length = 1.

When **istat** = -1, then on *input*, **vnoise** is the variance noise spectrum. NEW: To fit selected regions within a file, set **vnoise[i] = noise-variance** in the desired regions, and **vnoise[i] = 0.0** elsewhere.

When **istat** = 1, then on *input*, **vnoise[0]** is the noise variance if *you* wish to specify a value, else set **vnoise[0] = 0.0** to signal **rzrfit** to calculate the noise variance.

When **istat** = 2, then on *input*, **vnoise[0]** is the data scale factor if *you* wish to specify a value, else set **vnoise[0] = 0.0** to signal **rzrfit** to calculate the scale factor.

NOTE: The scale factor is the value by which you would multiply your data to obtain true counts. For example, suppose your data is meant to represent number of photons received at your counter. If you count for 10 seconds, and then divide the number of counts by 10 to express your data as counts/sec, you would need to multiply your data by a scale factor of 10 to transform it back to true counts. Since Poisson noise statistics require that the data be expressed in units of *counts*, (not counts/sec, not averaged counts/scan, etc.), the scale factor is important. If you do not know the scale factor of your data, set **vnoise[0] = 0.0**, and **rzrfit** will do the scaling for you.

baslin is an array with length = **n2+1** if the baseline **type** = 200, else length = 1.

When **type** = 200, then on *input*, **baslin** must be filled with the baseline spectrum.

ALSO SEE THE DISCUSSION BELOW FOR **type**.

w is a work array of length **n**. **w** is used in the fast mode, when **ifast** = 1, and in the bunch mode, when **nbunch** > 0. When **ifast** = 0 *and* **nbunch** = 0, then **w** may be a dummy array of length = 1. (Bunch processing is discussed on Page 101).

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

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

n = rzsizn(n2,datapeak,nl2)

ifast is an *input* flag which controls the processing mode. When **ifast** = 0, **rzrfit** uses the standard Levenberg-Marquardt processing method. When **ifast** = 1, a (faster) modified Levenberg-Marquardt mode is used for 60 – 90% of the iteration sequence. The standard Levenberg-Marquardt is *always* used for the final (2 or more) iterations.

When **ifast** = 0, **rzrfit** will fill the output array **yout** with the best-fitting model after each iteration. When **ifast** = 1, **yout** is only filled during the final cleanup step, when the covariance matrix and standard errors are also calculated.

When **ifast** = 0, **rzrfit** estimates the RMS noise using a running mean on the data. When **ifast** = 1, **rzrfit** uses the array **w** and the peakshape **datapeak** to get a better estimate of the RMS noise. Thus it is better to use **ifast = 1** if possible, because both the RMS noise and the chi-squared values will be better. Use **ifast = 0** when your problem is reluctant to converge.

istat is an *input* flag which describes the noise statistics. Set **istat** = 1 when the noise statistics are Normal, and the RMS noise, or the noise variance, is the same value everywhere in the interval **(0,n2)**. Set **istat** = 2 when the noise statistics are Poisson. Set **istat** = -1 if you wish to input your own noise variances in the array **vnoise**.

datmat is the parameter *input and output* matrix. See below.

covar is an array which will be used to calculate the covariance matrix for the parameters. The array length must be $\geq$ [**mmax*mmax**].

covar is *output* as the covariance matrix of the fit.

hess is a work array, of minimum length [**mmax*mmax**].

iwork is an integer work array, of minimum length 6***mmax**.

work is a float work array, of minimum length 9***mmax**.

mmax is *input*. It is the maximum number of parameters needed to describe the model. Most peaks, and most baselines, require 3 or 4 parameters each. **rzrfit** allows you to define your own peaks and baselines with as many as 6 parameters each. However, unless you are using your own peakshape functions, you will be safe if you set **mmax = 6*npks** for All-at-once processing, and **mmax = 6*nbunch** for Bunch-mode processing.

The parameters which must be entered into the data matrix **datmat[npks+2][40]**, are defined below.

datmat[npks+2][40]

An input/output matrix of peak parameters
First row is a control vector.
Last row is a control vector.
Other rows of **datmat** each contain parameters for one peak.
All quantities are *input* as *float numbers*.

Input parameters for first row of **datmat**:
datmat[0] = (npks, nbunch, bunchflag, 0, 0, 0, 0, 0, 0, 0, 0, xstart, xstep, 0...)

Output parameters for first row of **datmat**:
datmat[0] = (npks, reserved, bunchflag, iter, chisq, reserved, cnvg, reserved, reserved, j1=first peak of current bunch, j2=last peak of current bunch, xstart, xstep, reserved....)

npks is the *input number of peaks in the model*, expressed as a floating point number. (If you are using this library with a 16-bit compiler on a PC, you are limited to npks $\leq$ 30. Yes, segments bite again.)

nbunch is *input* as the *maximum number of peaks to be used in each bunch*, for bunch processing. (If you are using this library with a 16-bit compiler on a PC, you are limited to nbunch $\leq$ 30.) **nbunch** should be set to the maximum number of peaks in any region where the peaks are heavily overlapped. (You will not obtain good results if you ask **rzrfit** to break a region of 10 heavily overlapped peaks into 2 bunches of 5 peaks each.) However, when choosing **nbunch**, remember that smaller is faster, *a lot faster!*

If you are processing in the All-at-once mode (the usual mode for all peak- fitting algorithms devised up to this time), set **nbunch**=0.

bunchflag is a *input initialization flag* that tells **rzrfit** whether you want All-at-once processing **bunchflag**=0, or Bunch-mode processing **bunchflag** = 1. If you initialize for Bunch-mode processing, monitor this flag during the iteration sequence. **rzrfit** will signal that it has finished with all peaks, all bunches, by setting **bunchflag = 0**.

iter is the iteration number. **It MUST be set to 0** the first time **rzrfit** is called. Thereafter, **rzrfit** will maintain **iter**. The value of iter will be increased each iteration until final convergence is reached. When **rzrfit** converges, it will automatically set the value of **iter** to -1.0. The value **iter = 0.0** is the signal that **rzrfit** is has converged, and has performed a final cleanup, calculating peak areas, the standard errors of the parameters, and the covariance matrix. If you find yourself in a situation where **rzrfit** has not yet converged, and you wish to force **rzrfit** to perform the final clean-up, set **iter = -1.0**. **rzrfit** will increment **iter** to zero when it is finished.

You should check **iter** after each iteration, to find out when **rzrfit** is finished (**iter**=0). If you have set the **bunchflag** for Bunch-mode processing, wait for **bunchflag**=0 && **iter**=0.

cnvg is *output*, the *convergence number* at the end of each iteration. When **cnvg**=5, **rzrfit** has converged to an answer, and it will set **iter**=-1 internally for the final clean-up pass.

chisq is *output*, the *reduced chi-squared value* at the end of the current iteration.

j1 is maintained by **rzrfit**. It indicates the first peak of the current bunch.

j2 is maintained by **rzrfit**. It indicates the last peak of the current bunch.

xstart, xstep are *input* as the *x-value corresponding to the first data point in* **ydata**, *and the x-interval between data points in* **ydata**. **xstart** and **xstep** are not normally

used by **rzrfit**. However, there may be circumstances where they may be useful to the programmer in the **rzrserve** function **rzupdt**, and so space has been reserved in **datmat** for them. You do not need to load **xstart, xstep** in **datmat** unless you also change **rzupdt** to use these parameters.

Input parameters for second (third, etc) row of **datmat**:
datmat[1] = (
type, c, fixc, h, fixh, w, fixw, a, fixa, p, fixp, q, fixq, 0, 0, 0, 0, 0, master/slave, 0, 0,lowlimc,highlimc,lowlimh,highlimh, lowlimw,highlimw,lowlima,highlima,lowlimq,highlimq, 0, ...)

Output parameters for second (third, etc) row of **datmat**:
datmat[1] = (type, c, errc, h, errh, w, errw, a, erra, p, errp, q, errq, area, errarea, reserved, ...)

type is a number which identifies the peak type:

0. Captured DataPeak, input in the **datapeak** array
1. gauss
2. lorentz
3. weighted sum of gauss and lorentz
4. product of gauss and lorentz
5. asymmetric gauss
6. asymmetric lorentz
7. Pearson VII
8. Log Normal

9.-10. Create your own peakshape. Follow the syntax shown in the function **rzpk1** of **rzrser02.for**. Fill either function rzpk9 or **rzpk10**, and assign type number 9 or 10 to the new peak. Recompile **rzrser02.for** after revision.

200. baseline stored in array **baslin**.
201. constant (offset) baseline
202. linear baseline
203. quadratic baseline
204. exponential baseline

205.-209. Roll your own baseline. See **rzrser02.for**.

c is an estimated peak **center** position, measured in data point numbers.

h is the estimated peak **height**.

w is the estimated peak **width** (or the third parameter — see Chapter 8).

a is the estimated fourth parameter (**asymmetry**, mixing, width, etc. — see Chapter 8).

p is estimated fifth **parameter**, if any.

q is estimated sixth **parameter**, if any.

fix identifies whether the parameter is to be varied during the current iteration of the fit.

fix < 0.0 for a parameter which is currently variable.
fix = 0.0 when a parameter is currently fixed (nonvariable).
fix > 0.0 *and* < 9.0 when a parameter is currently variable, and is also constrained to be positive. (See function **rzlims** in **rzrser02.for**.)

fix = 9.0 when a parameter is currently variable, but is also constrained to lie between lowlim and highlim as entered into the same row of datmat. Note that the lowlim and highlim values in datmat are **only** used when **fix** = 9.

fix > 10.0 *and* < 100.0 when a parameter is currently variable, but is also constrained to lie within **fix%** of the starting value given in datmat.

fix = 100.0 is a signal that the corresponding parameter partakes in the master/slave relationship. *Do not use fix = 100.0 unless the current peak is a slave, and the master/slave relationship has been established in the 19th column of the current row!* When **fixc** = **100.0**, then the corresponding parameter **c** is interpreted as the constant-offset of this slave peak from its master. When **fixh** = **100.0**, then the corresponding parameter **h** is interpreted as the constant-ratio of slave peak-height to master peak-height. When **fixw** = **100.0**, then the corresponding parameter **w** is interpreted as the constant-ratio of slave peak-width to master peak-width. When **fixa** = **100.0**, then the corresponding parameter **a** is interpreted as the constant-ratio of slave peak-asymmetry to master peak-asymmetry. When **fixp** = **100.0**, then the corresponding parameter **p** is interpreted as the constant-ratio of slave parameter-p to master parameter-p.

We also recommend setting **fix** = 0.0 at all unused parameter locations within **datmat**. (For example, when **type** = 1, for a gaussian peak, only the parameters **c**, **h**, and **w** are used. However, setting **fixa** = 0.0, and **fixp** = 0.0 could save you some grief later on, if you decide to alter the function **limits** in **rzrserve**.

area, errarea are unused on input.

master/slave An odd integer indicates a master; odd+1 indicates corresponding slave. Thus the first master peak will contain a 1 in the master/slave column; all its slaves will contain a 2 in the master/slave column. The second master will contain a 3 in the master/slave column, and all its slaves will be numbered 4, etc..

Put a 0 (zero) in the master/slave column when peaks do not partake in a master/slave relationship. Always put a zero in the master/slave column if you are using bunch-mode processing! **Bunch processing is not compatible with master/slave processing.**

Note that **rzrfit** will resort the rows when any master/slave relationships are established. This is necessary because the processing algorithm requires that slaves be in rows following the row of the master. YOU do not need to follow this rule in setting up **datmat**, however.

The example on page 105 shows how to use HANDLE to set up the master/slave relationship. Look at the HANFIL function in handle2.for for programmer instructions on creating master/slave relationships.

lowlimc, highlimc are upper and lower limits placed on the values for the parameter c. Note: **lowlimc, highlimc** are **only** used if **fixc** is set to 9. For all other values of **fixc**, lowlimc and highlimc will be ignored.

lowlimh, highlimh, lowlimw, ... are upper and lower limits placed on the parameters h, w, etc.

We have provided a function in **handle.c** named **rzrfil**. It is intended to assist the programmer in filling the **datmat** matrix. It is well-documented in **handle.c**.

At the end of each iteration, **datmat** contains the updated values of each parameter value, where appropriate.

Upon return from the final iteration, after convergence is achieved, each **fix** value is replaced by the **standard error of the corresponding parameter**. The peak area, or the area under the baseline, and its standard error, fill the last two spaces.

NOTE: If the parameter was fixed (**fix** = 0), then the value returned in the error position is zero, as a reminder that the parameter was not varied. When a returned standard error is -1, this indicates that the diagonal element of the covariance matrix was negative, and the square root could not be taken. A negative diagonal element is an indication that the fit has not really converged, i.e., the solution is not sitting in a minimum in parameter space.

7.3 First Example using rzrfit

First, we used the **rzrpic** peak picker on our spectrum, exactly as shown in Chapter 6. Then we called upon **rzrfit**, proceeding as shown below.

We chose to fit a model consisting of five Gaussian peaks, plus a linear baseline. Note that once we selected our model, all the initial parameters were estimated automatically within **handle** by **rzrfil**. Furthermore, with these good initial estimates, **rzrfit** converged in 4 iterations.

Data file: SPEC2

Model chosen: 5 Gaussian peaks, plus a linear baseline. All the initial parameters were estimated by handle and HANFIL, freeing the user from this onerous task. We will link two of the peaks in a master/slave relationship, to show you how to set this up. (Our master/slave link, and the choices for the parameters, is purely arbitrary).

Notice that RZRFIT resorts the rows in DATMAT. This is necessary, because RZRFIT wants slaves to be placed in rows immediately following the row of their master.

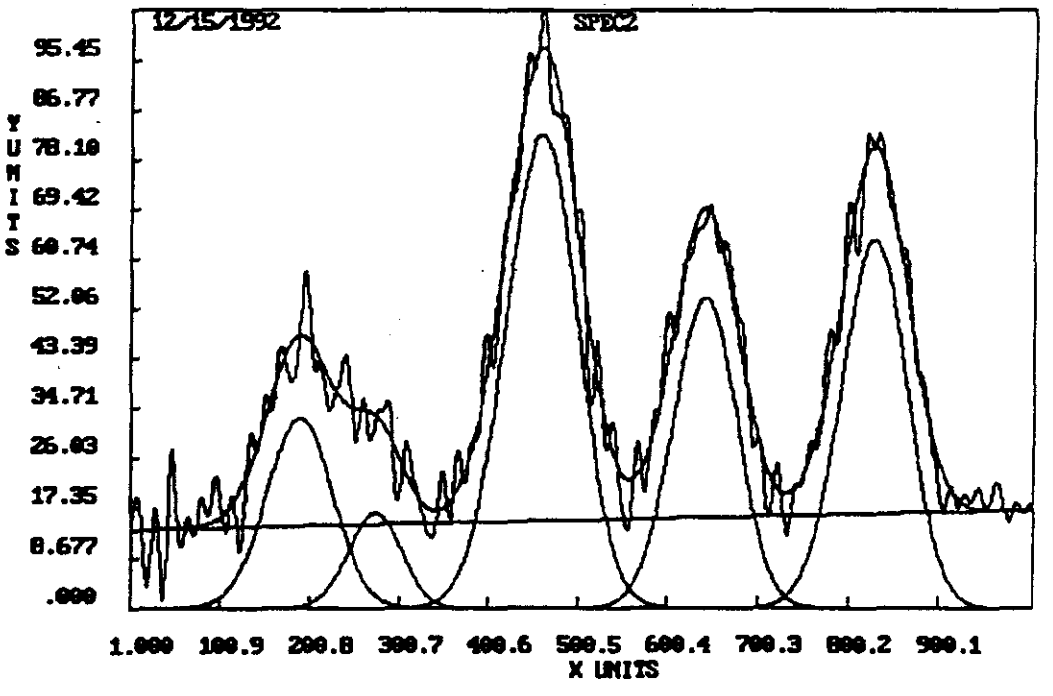
Using handle:

```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
  Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
.....................
.....................
.....................
Choose an operation: fit

Handle loads FIT from PIC data, so you must
run PIC or BAS IMMEDIATELY BEFORE THIS FIT.
If you did not, type R to Restart.
Did you run PIC or BAS? [Y] y

Bunch-processing is faster than processing the peaks All-at-once.
Process the peaks in small bunches? Y/N [N] n
Identify the type of your peaks
  Type number     Shape
     0        DataPeak [default]
     1        Gaussian
     2        Lorentzian
     3        Sum(Gaussian+Lorentzian)
     4        Product(Gaussian*Lorentzian)
     5        Asymmetric Gaussian
     6        Asymmetric Lorentzian
```

```
     7        Pearson 7
     8        Log Normal
     100       1st & 2nd DataPeaks
Enter type number: 1

Will there be a baseline?: [Y] y
Available baseline types are
   O = Offset
   L = Linear
   Q = Quadratic (not recommended)
   E = Exponential
   F = File Baseline
Select Baseline Type [L] : L

Here is the control vector, DATMAT(0):
 npks nbunch  bnchflg  iter  chisq chitest cnvg
  6.00   0.00   0.00   0.00   0.00   0.00   0.00
DATMAT(i), the input data matrix, starting i=1:
Row Type  Cent  Fix  Hght  Fix  Wdth  Fix  Aparm Fix  Master/Slave
 2  1  450.00 -3.00 82.58  1.00 77.44  3.00  0.00  0.00 0
 3  1  829.00 -3.00 62.74  1.00 99.39  3.00  0.00  0.00 0
 4  1  653.00 -3.00 49.69  1.00 71.33  3.00  0.00  0.00 0
 5  1  177.00 -3.00 32.81  1.00 89.83  3.00  0.00  0.00 0
 6  1  283.00 -3.00 17.19  1.00 58.75  3.00  0.00  0.00 0
 7 202   1.41 -1.00  0.00 -1.00  0.00  0.00  0.00  0.00 0

Do you wish to link peaks? [N] : y
A group of Linked peaks must refer to a Master peak.
Enter Master peak ROW: (N for none) : 2
List Slave peaks by ROW: (N for none) : 4
Slave in ROW:  4.  For each parameter,
 either enter the slaving value, or enter 'N' if Not-slaved.
Position: Enter fixed offset of slave peak: 200
Amplitude: Enter slave/master amp ratio: .65
Width: Enter slave/master width ratio: 1
Asymmetry: Enter slave/master asym ratio: n
List Slave peaks by ROW: (N for none) : n

Here is the control vector, DATMAT(0):
 npks nbunch  bnchflg  iter  chisq chitest cnvg
  6.00   0.00   0.00   0.00   0.00   0.00   0.00
DATMAT(i), the input data matrix, starting i=1:
```

```
Row Type  Cent  Fix  Hght  Fix  Wdth  Fix  Aparm Fix  Master/Slave
 2   1 450.00 -3.00 82.58  1.00 77.44  3.00  0.00  0.00 1
 3   1 829.00 -3.00 62.74  1.00 99.39  3.00  0.00  0.00 0
 4   1 200.00 100.00  0.65 100.00  1.00 100.00  0.00  0.00 2
 5   1 177.00 -3.00 32.81  1.00 89.83  3.00  0.00  0.00 0
 6   1 283.00 -3.00 17.19  1.00 58.75  3.00  0.00  0.00 0
 7 202   1.41 -1.00  0.00 -1.00  0.00  0.00  0.00  0.00 0

Do you wish to link peaks? [N] : n

The default (and most usual) type is Normal noise.
Enter N for Normal noise; P for Poisson noise [N] : n
Enter Normal noise variance.  Enter 0 if unknown: 0
Select F or S, Fast or Standard mode: [F]         f
Entering RZRFIT with iter=0.  Wait for setup...
Processing peaks 1 to 6 of 6.  Noise variance=11.6199
 Reduced chisq=23.57  at iter 1
 ........
 Reduced chisq=6.54  at iter 4
 .......
 Reduced chisq=3.063  at iter 14
 Reduced chisq=3.063  at iter 0

                OUTPUT DATA MATRIX
Pk Type Cent   Err    Hght Err    Wdth Err   Aparm  Err   Area  Err
1 1 187.07  3.1  32.33  1.1  82.27  6.2   0.00  0.0 2831.31 233.9
2 1 269.08  5.1  16.56  1.5  67.68  9.0   0.00  0.0 1193.27 192.6
3 1 449.39  0.4  79.51  0.7  92.82  1.1   0.00  0.0 7856.15 119.3
4 1 649.39  0.0  51.68  0.0  92.82  0.0   0.00  0.0 5106.50  0.0
5 1 822.74  0.6  65.09  1.0  83.59  1.6   0.00  0.0 5792.17 143.3
6 202  14.59  0.6   0.00  0.0   0.00  0.0   0.00  0.0 15320.28 767.4
 Press ¡Enter¿ to see datmat in user coordinates ...

          OUTPUT DATA MATRIX in user coordinates
Pk Type Cent   Err    Hght Err    Wdth Err   Aparm  Err   Area  Err
1 1 188.07  3.1  32.33  1.1  82.27  6.2   0.00  0.0 2831.31 233.9
..................
Reduced chisq (whole file) =3.06288
Variance = 0.1162E02
Processed with istat=1 (1=Normal noise statistics, 2=Poisson)
RESULT MAY BE SAVED TO A FILE
```

```
Press ENTER to return to menu.
```



7.4 Second Example using rzrpic and rzrfit

rzrfit lets you fit *real data* shapes to your data. We will illustrate this using radiochromatography flow-cell peaks.

FLOWCAL is a *real data* peak, which we have smoothed. It was obtained during a radiochromatography run. The detector was counting scintillations in a flow cell.

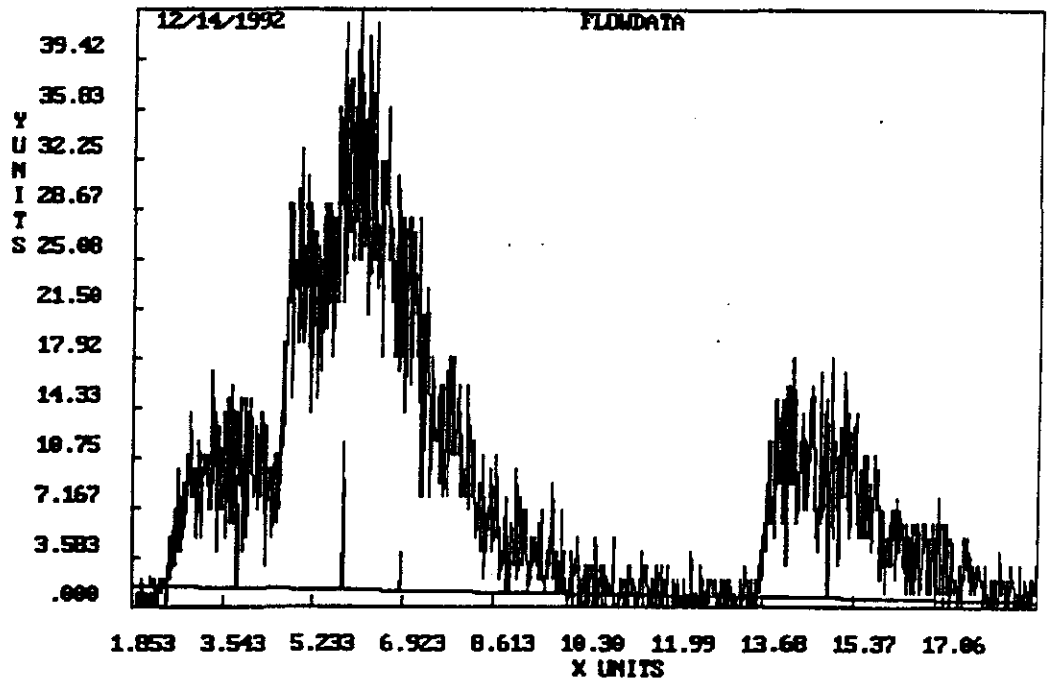
FLOWDATA is simulated data. We added together four peaks just like **FLOWCAL**. Three of the peaks have maximum counts of 10; one has a maximum count of 20. We also added a small background count. Then we put the appropriate counting noise (Poisson) on the simulated data. **FLOWDATA** is the result.

We will run **rzrpic** on the data, and accept the biggest 8 peaks, pretending that we do not know how many peaks are really there. The results will show the 4 peaks which are really there, and their heights will be correct within 2 standard deviations. The 4 nuisance peaks which we will accept will show up as much smaller; two of them will be consistent with peak heights = 0.

Data file: FLOWDATA
Captured DataPeak: FLOWCAL
Using HANDLE:

```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
  Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
.............
.............
.............
Choose an operation: pic

Enter name of spectrum (Try SPEC2): flowdata
Enter name of peakshape (Try PEAK2): flowcal

Enter -1 for Quick-Pick
Enter  1 for High-Performance picker
Enter  2 for High-Resolution picker
Enter  3 for 2nd-Order High-Performance picker
Enter  4 for Quiet Pick (High Res. Good for very narrow peaks)
 Select picker [2]: 2
Enter peak threshold (number of standard deviations of noise)
Enter 3 if unsure: -3
Enter N for Normal noise; P for Poisson noise [N]: p

Entering RZRPIC. Please wait for processing...

Peaks found by RZRPIC.
```

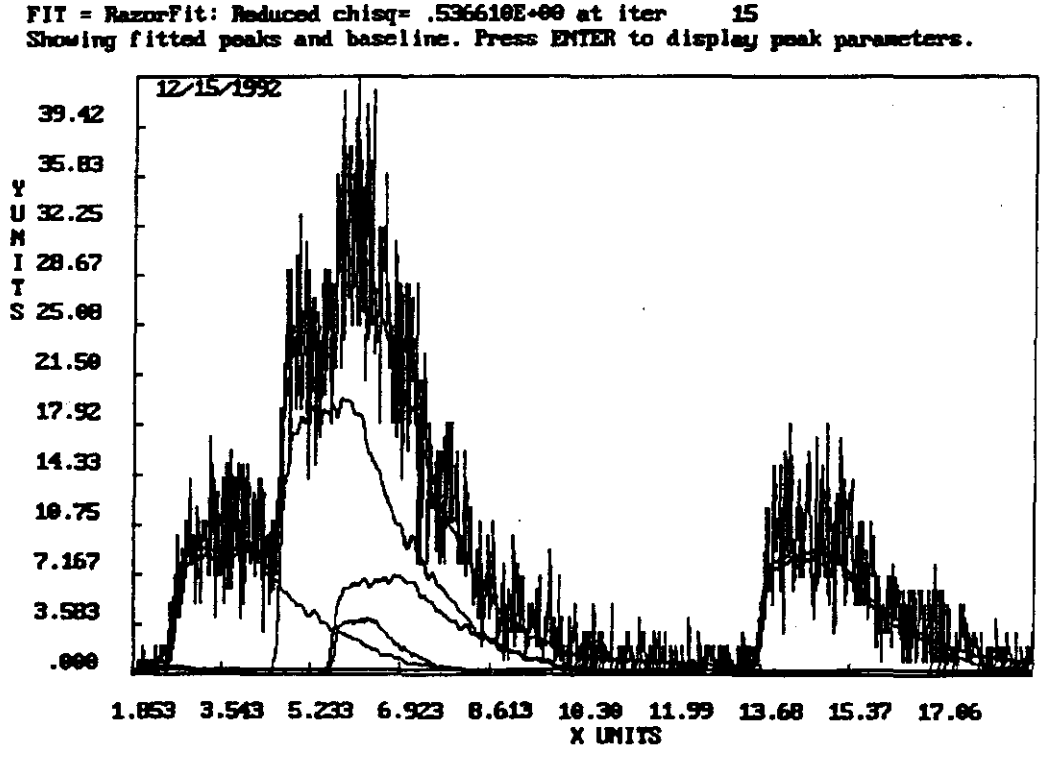
```
Estimated RMS noise: 2.89173
Number Peaks detected: 7
Using Peak Detection Threshold: -3
Peak AREA Significances:
 19.87  15.31  11.88   5.41   4.43   3.94   3.61
Number peaks (as sorted) accepted for FIT: 7
Select:
Print (S)ignificances, (L)ocations, (H)eights, (W)idths, (E)verything.
(R)esort. (T)uneup heights. (A)ccept. (M)enu.
m

    Press ENTER to return to menu.
```



```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
  Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
....
....
....
Choose an operation: fit

Handle loads FIT from PIC data, so you must
run PIC or BAS IMMEDIATELY BEFORE THIS FIT.
If you did not, type R to Restart.
Did you run PIC or BAS? [Y] y
Bunch-processing is faster than processing the peaks All-at-once.
Process the peaks in small bunches? Y/N [N] n
Identify the type of your peaks
  Type number     Shape
     0        DataPeak [default]
     1        Gaussian
     2        Lorentzian
     3        Sum(Gaussian+Lorentzian)
     4        Product(Gaussian*Lorentzian)
     5        Asymmetric Gaussian
     6        Asymmetric Lorentzian
     7        Pearson 7
     8        Log Normal
     100       1st & 2nd DataPeaks
Enter type number: 0
Preserve or Vary the DataPeak width? P/V [P]: p
Preserve or Vary the DataPeak asymmetry? P/V [P]: p

Will there be a baseline?: [Y] y
Available baseline types are
   O = Offset
   L = Linear
   Q = Quadratic (not recommended)
   E = Exponential
   F = File Baseline
Select Baseline Type [L]: o

Here is the control vector, DATMAT(0):
 npks nbunch  bnchflg  iter  chisq chitest cnvg
 8.00   0.00   0.00   0.00   0.00   0.00   0.00
DATMAT(i), the input data matrix, starting i=1:
```

```
Row Type  Cent   Fix  Hght  Fix  Wdth  Fix  Aparm Fix  Master/Slave
 2   0 777.00 -3.00  9.44  1.00 71.95  0.00  0.00  0.00 0
 3   0 116.00 -3.00  8.05  1.00 65.87  0.00  0.00  0.00 0
 4   0 236.00 -3.00 10.66  1.00 67.59  0.00  0.00  0.00 0
 5   0 953.00 -3.00  1.08  1.00 69.00  0.00  0.00  0.00 0
 6   0 506.00 -3.00  0.91  1.00 64.69  0.00  0.00  0.00 0
 7   0 420.00 -3.00  1.58  1.00 44.65  0.00  0.00  0.00 0
 8   0 300.00 -3.00  2.77  1.00 58.63  0.00  0.00  0.00 0
 9 201   0.00 -1.00  0.00  0.00  0.00  0.00  0.00  0.00 0
Do you wish to link peaks? [N] : n
The default (and most usual) type is Normal noise.
Enter N for Normal noise; P for Poisson noise [N] : p
Enter Poisson scaling factor.  Enter 0 if unknown: 0
Select F or S, Fast or Standard mode: [F] f
Entering RZRFIT with iter=0.  Wait for setup...
Processing peaks 1 to 8 of 8.  Noise variance=0.602189
 Reduced chisq=3.481  at iter 1
 Reduced chisq=1.674  at iter 2
 ............

Reduced chisq=0.549  at iter 15
 Reduced chisq=0.546  at iter 16
 Reduced chisq=0.546  at iter 17
 Reduced chisq=0.546  at iter 0

                OUTPUT DATA MATRIX
Pk Type Cent  Err   Hght Err   Wdth Err   Aparm  Err   Area  Err
1 0 775.27  0.7   8.46  0.2 148.00  0.0   0.08  0.0 1291.24 32.1
2 0 116.95  0.8   9.09  0.2 148.00  0.0   0.08  0.0 1386.86 36.6
3 0 236.56  0.6  20.88  0.6 148.00  0.0   0.08  0.0 3188.05 84.7
4 0 -1014.00  0.0   0.00  0.0 148.00  0.0   0.08  0.0    0.01  0.0
5 0 -1014.00  0.0   0.00  0.0 148.00  0.0   0.08  0.0    0.25  0.0
6 0 -1014.00  0.0   0.00  0.0 148.00  0.0   0.08  0.0    0.21  0.0
7 0 294.98  1.8   8.10  0.5 148.00  0.0   0.08  0.0 1237.12 76.1
8 201   0.55  0.1   0.00  0.0   0.00  0.0   0.00  0.0  556.57 54.1
 Press ¡Enter¿ to see datmat in user coordinates ...

          OUTPUT DATA MATRIX in user coordinates
Pk Type Cent  Err   Hght Err   Wdth Err   Aparm  Err   Area  Err
1 0  14.77  0.0   8.46  0.2   0.00  0.0   0.08  0.0    0.00  0.0
2 0   3.80  0.0   9.08  0.2   0.00  0.0   0.08  0.0    0.00  0.0
3 0   5.80  0.0  20.88  0.6   0.00  0.0   0.08  0.0    0.00  0.0
```

```
4  0  -15.05  0.0   0.00  0.0   0.00  0.0   0.08  0.0   0.00  0.0
5  0  -15.05  0.0   0.00  0.0   0.00  0.0   0.08  0.0   0.00  0.0
6  0  -15.05  0.0   0.00  0.0   0.00  0.0   0.08  0.0   0.00  0.0
7  0    6.77  0.0   8.10  0.5   0.00  0.0   0.08  0.0   0.00  0.0
8 201   0.55  0.1   0.00  0.0   0.00  0.0   0.00  0.0   0.00  0.0
Reduced chisq (whole file) =0.546
Processed with istat=2 (1=Normal noise statistics, 2=Poisson)
RESULT MAY BE SAVED TO A FILE

Press ENTER to return to menu.
```



7.5 Third Example using rzrbas and rzrfit

rzrfit will process peaks in small 'bunches', allowing you to process more than 30 peaks at a time (the DOS limit), and giving you the results *a lot faster*! In addition, **rzrfit** knows how to use data with Poisson noise. We illustrate these features using x-ray diffraction data.

XRAYSCAN contains x-ray diffraction peaks, measured for diffraction angles between 10 and 90 degrees. The noise in the data is somewhere between Normal and Poisson. (It would be purely Poisson were it not for instrumental effects.) We will assume the noise is Poisson, to illustrate some things you need to think about when dealing with Poisson noise.

XRAYPEAK is a synthetic Gaussian peak, approximately the same width as the peaks in XRAYSCAN.

Think about these things when attempting peak-fitting on data with Poisson noise.

- When the noise is Poisson, the RMS noise is *larger* on the peaks, and *smaller* in the baseline regions. **rzrfit** knows that Poisson noise is *proportional to* the square-root of the signal, and computes a proper value of Chisq using that knowledge.

  NOTE: We said *proportional to*, not equal to the square-root. Thus you may rescale your Poisson data without penalty. The data in XRAYSCAN was renormalized from counts to counts/sec. **rzrfit** performs peak-fitting correctly on this normalized data.

- If you remove a baseline, or a background, from data that has Poisson noise, the resultant background-corrected data no longer has Poisson noise.

  NOTE: When you remove a baseline, the noise is no longer *proportional to* the square-root of the signal. Thus you may freely *multiply* your Poisson data by a constant factor, but *do not subtract* a background.

  rzrfit allows you to *define the baseline or background* you want, and enter it into the **baslin** array. When you do this, it will fit any peaks on top of your baseline, or background. The Poisson-nature of the noise will be correctly handled.

We will run **rzrbas** on the data, in order to find peaks and get a baseline scan to use for background correction. We will then save the baseline into a file named XRAYBASE.

The baseline file XRAYBASE will be used to get the correct answer for Poisson noise statistics.

Incidently, if you want to have a baseline when you use **rzrfit** in the 'Bunch' mode, you MUST have a baseline file. Handle has been programmed so that 'file' is your only choice for a baseline during Bunch-processing.

We will use the 'Quiet-Pick' peak picker, because the peaks in XRAYSCAN are approximately 2 datapoints wide - too narrow for good results with any of the other pickers except 'Quick-Pick'. However, 'Quick-Pick' is unable to resolve the xray peak components.

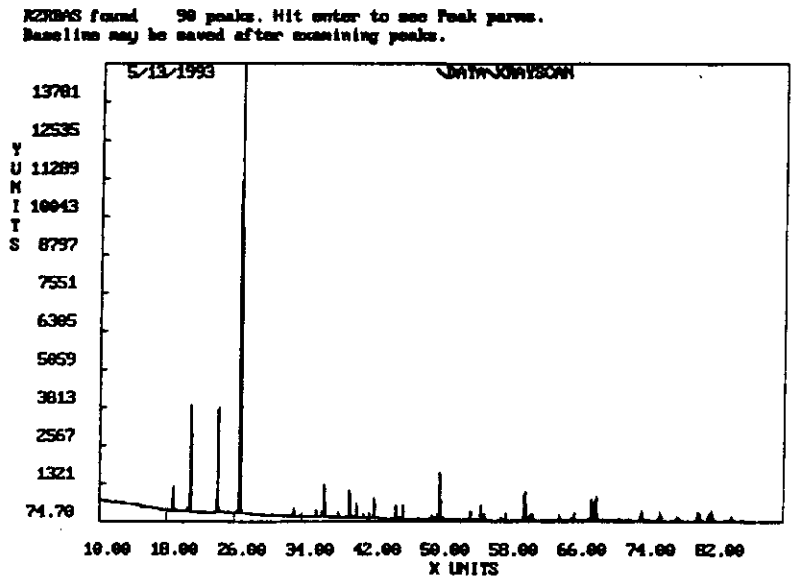
We will also resort the peaks by height, to illustrate the use of **rzpkst** (See page 175), and then accept only the 75 largest peaks for FIT.

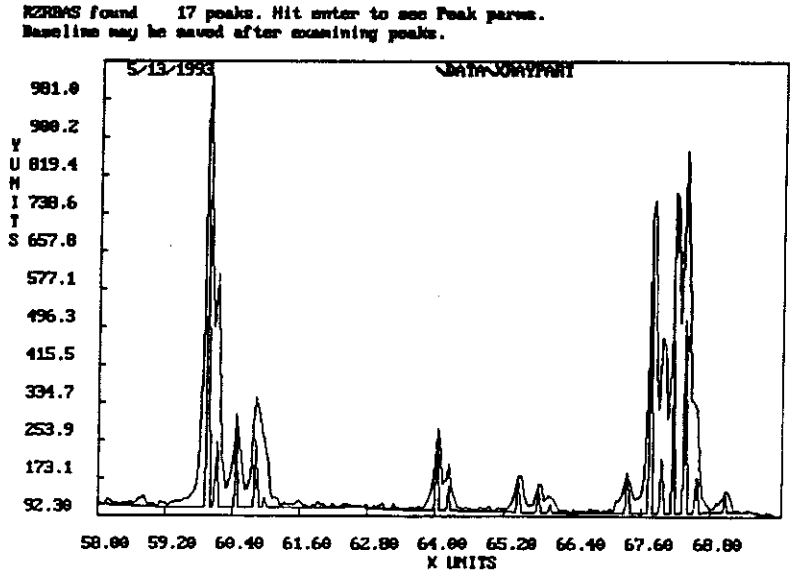
Data file: XRAYSCAN
Peakshape file: XRAYPEAK
Using HANDLE:

```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
  Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
....
....
Choose an operation: BAS

Enter name of spectrum (Try SPEC8): XRAYSCAN
Enter name of peakshape (Try PEAK8): XRAYPEAK
RZRBAS also picks peaks! Choose picker.
Enter -1 for Quick-Pick
Enter 1 for High-Performance picker
Enter 2 for High-Resolution picker
Enter 3 for 2nd-Order High-Performance picker
Enter 4 for Quiet Pick (High Res. Good for very narrow peaks)
 Select picker by Number [2]: 4
Enter peak threshold (number of standard deviations of noise): 3
Enter N for Normal noise; P for Poisson noise [N]: P

Entering RZRBAS with bsens = 1. Wait for setup...
Estimated RMS noise: 19.86
Using Peak detection threshold:  3.0
# Peaks detected: 94

Peak Start/stop regions:
  0/ 148  148/ 251  251/ 497  497/ 818  818/ 981  981/1600
Current baseline sens, BSENS = 1
Enter new value for BSENS (Enter 0 to quit):
BASELINE MAY BE SAVED TO A FILE AFTER EXAMINING PEAKS.
Hit any key to examine peak parameters.


Peaks found by RZRBAS.
Estimated RMS noise: 19.86
Number Peaks detected: 94
Using Peak Detection Threshold: 3
```

```
Peak HEIGHT Significances:
.....
Number peaks (as sorted) accepted for FIT: 94
Select:
Print (S)ignificances, (L)ocations, (H)eights, (W)idths, (E)verything.
(R)esort. (T)uneup heights. (A)ccept. (M)enu.
R
Resort by (S)ignificances, (L)ocations, (H)eights, (W)idths? [L]
H

Peaks found by RZRBAS.
Estimated RMS noise: 19.86
Number Peaks detected: 94
Using Peak Detection Threshold: 3
Peak Heights:
....
Number peaks (as sorted) accepted for FIT: 90
Select:
Print (S)ignificances, (L)ocations, (H)eights, (W)idths, (E)verything.
(R)esort. (T)uneup heights. (A)ccept. (M)enu.
A
# Peaks detected: 94
How many do you accept for FIT?
75

Peaks found by RZRBAS.
....
Select:
Print (S)ignificances, (L)ocations, (H)eights, (W)idths, (E)verything.
(R)esort. (T)uneup heights. (A)ccept. (M)enu.
M

Press ENTER to return to menu.
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
......
......
.....
Choose an operation: SAV

Under what name?
XRAYBASE
```

RZRBAS has given us a set of peaks, and a baseline, for use in **RZRFIT**. The results are shown below:





We proceed directly from **RZRBAS** into **RZRFIT**. Handle uses the function **RZDFIL** (See page 177) to fill the **datmat** array with the peak locations, heights, and widths that are always provided by **RZRBAS** and **RZRPIC**.

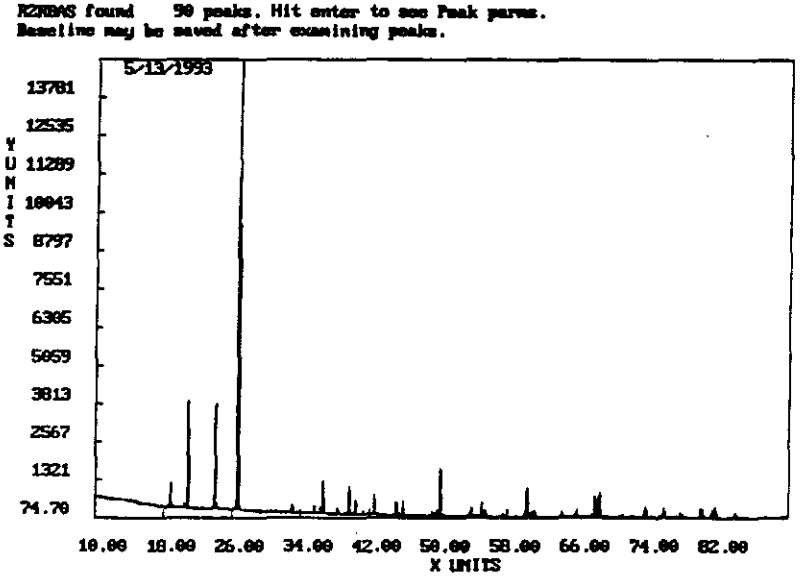
When running **RZRFIT**, you need to decide whether to run in Fast or Standard mode (Fast can be significantly faster; Standard is a standard Levenburg- Marquardt available anywhere). You also will specify whether to use All- at-once or Bunch-mode processing. With $\geq$ 30 peaks, under DOS, only Bunch-mode is possible. And, it's a lot faster.

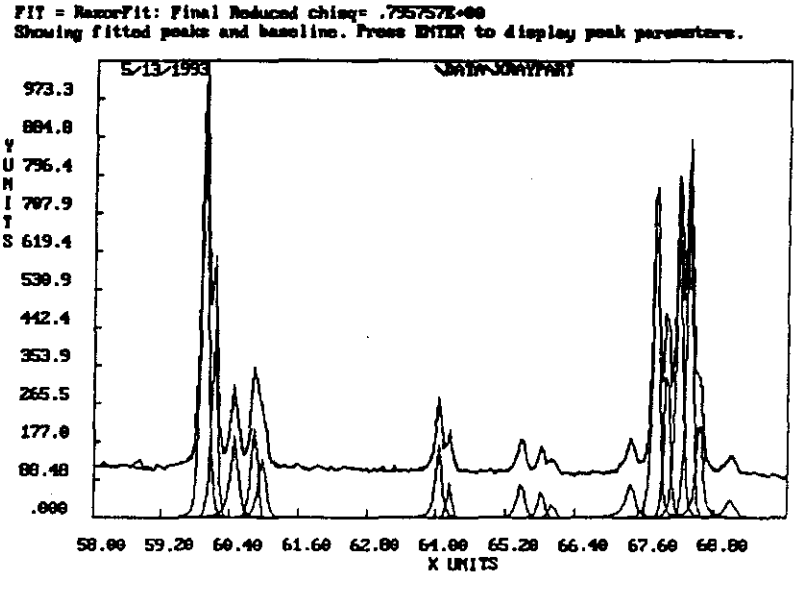
```
RAZOR LIBRARY for Spectral Analysis -¿ There is only one best way!
  Maximum Likelihood (ML), Maximum Entropy (ME), and Bayesian processing.
..........
Choose an operation: FIT

Handle loads FIT from PIC data, so you must
run PIC or BAS IMMEDIATELY BEFORE THIS FIT.
If you did not, type R to Restart.
Did you run PIC or BAS? [Y] Y

Bunch-processing is faster than processing the peaks All-at-once.
Process the peaks in small bunches? Y/N [N] y
Enter maximum number of peaks in a bunch [Suggest 4 or 5] : 5
Identify the type of your peaks
  Type number     Shape
     0        DataPeak [default]
     1        Gaussian
     2        Lorentzian
     3        Sum(Gaussian+Lorentzian)
     4        Product(Gaussian*Lorentzian)
     5        Asymmetric Gaussian
     6        Asymmetric Lorentzian
     7        Pearson 7
     8        Log Normal
     100       1st & 2nd DataPeaks
Enter type number: 1
Will there be a baseline? [Y] : Y

Enter name of baseline file, else ¡Enter¿ to use PIC/BAS basln:
 XRAYBASE

Here is the control vector, DATMAT(0):
 npks nbunch  bnchflg  iter  chisq chitest cnvg
 76.00  5.00   1.00   0.00   0.00   0.00   0.00
DATMAT(i), the input data matrix, starting i=1:
```

```
Row Type  Cent  Fix  Hght  Fix  Wdth  Fix  Aparm Fix  Master/Slave
 2  1 167.00 -4.00 32.16  1.00  2.62  3.00  0.00  0.00 0
 3  1 175.00 -4.00 747.51  1.00  3.56  3.00  0.00  0.00 0
....
...
75  1 1548.00 -4.00 30.78  1.00  3.80  3.00  0.00  0.00 0
76  1 1588.00 -4.00 25.17  1.00  5.08  3.00  0.00  0.00 0
77 200   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 0
Do you wish to link peaks? [N] : N
The default (and most usual) type is Normal noise.
Enter N for Normal noise; P for Poisson noise [N] : P
Select F or S, Fast or Standard mode: [F] F

Entering RZRFIT with iter=0.  Wait for setup...
Processing peaks 1 to 4.  Noise variance=9.10942
 Reduced chisq=1058.82  at iter 1
.................
Processing peaks 1 to 4 of 76
.................
Processing peaks 5 to 7 of 76.
.................
Processing peaks 8 to 8 of 76.
.................
.................
Processing peaks 74 to 75 of 76.

.........................

          OUTPUT DATA MATRIX in user coordinates
Pk Type Cent   Err    Hght Err    Wdth Err   Aparm  Err   Area  Err
1 1   19.01  0.0   15.93 29.6    0.00  0.0   0.00  0.0    0.00  0.0
2 1   18.77  0.0  746.58 125.0   0.00  0.0   0.00  0.0    0.00  0.0
.........................
75 1   89.44  0.0   19.44  7.5    0.00  0.0   0.00  0.0    0.00  0.0
76 200   0.00  0.0   0.00  0.0    0.00  0.0   0.00  0.0    0.00  0.0
Reduced chisq (whole file) =15.896
Processed with istat=2 (1=Normal noise statistics, 2=Poisson)
RESULT MAY BE SAVED TO A FILE
```

If you run a portion of XRAYSCAN using the Razor Library DEMO, your final screen

will look like the picture below.





7.6 The RazorFit algorithm

In the final sections of this chapter, we will (1) present the assumptions that all peak-fitting methods make about your data, and carefully discuss the assumptions about your noise, or data errors, that you implicitly make when using a program like RazorFit, (2) derive the RazorFit algorithm from the Maximum Likelihood principle, and (3) discuss mathematical details.

7.7 The RazorFit model

When you use RazorFit, or similar peak-fitting programs, you are assuming your data can be fit by a model. The model usually can be described with an analytical expression, although it need not be. The important characteristic of the model is that whereas it has M parameters whose values are not known, M is a number smaller than the number of data points.

RazorFit uses this model for your data:

$$\mathbf{model(x)} = \sum_{\mathbf{j=1}}^{\mathbf{J}} \mathbf{(peak_j(x))} + \mathbf{baseline(x)},$$

where J is the number of peaks being fit, and each $peak_j(x)$ is synthesized using one of the analytical expressions given in Chapter 8. The function $baseline(x)$ is also given in that chapter. The number of parameters M is the sum of the parameters required for each peak, plus any baseline parameters.

RazorFit is the appropriate peak-fitting algorithm to use when your data errors are random, and additive, (Normal noise), and also when your data errors are the result of counting statistics (Poisson noise). Mathematically, RazorFit is appropriate when

$$\mathbf{data(x)} = \mathbf{model(x)} + \mathbf{error(x)},$$

when $error(x)$ has an equal chance of being a positive or a negative value, and when $error(x_i)$ is uncorrelated with $error(x_j)$ for all i, j.

Visually, the assumptions about $error(x)$ are represented in the figure below. The figure is a histogram of the errors measured at a *single* position x_k,

$$\mathbf{error(x_k)} = \mathbf{data(x_k)} - \mathbf{model(x_k)},$$

for a large number of measurements, on one sample, at position x_k.

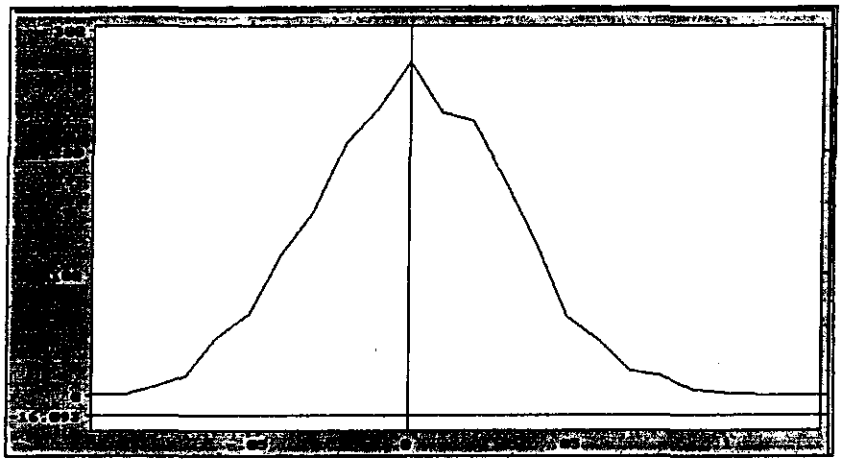


Fig 7.7.1 Histogram of errors measured at x_k.

The important characteristics of the data errors which are represented by the histogram are:

- The errors are nearly equally distributed between positive and negative values.
- Small errors are more probable than large ones. The histogram is peaked at zero.
- The RMS error need not be the same for all x. A histogram of errors measured at *any* data point x_j would have the same general shape, but not necessarily the same width, as the one above.

If a histogram of your data errors, measured at any point in your data set, would look similar to the one shown above, then RazorFit is an appropriate peak-fitting algorithm.

RazorFit provides solutions for data with the following types of noise statistics:

- Normal (gaussian-distributed) noise errors, with constant variance.
- Normal (gaussian-distributed) noise errors, with variance that depends on position within the data file.
- Poisson (counting statistics) noise errors, with variance proportional to the square-root of the signal.

7.8 RazorFit and Maximum Likelihood

Suppose you know that your data set, in the absence of noise or sampling errors, could be described by a model, $model(x)$. This model is known when its M parameters are known. You have measured N data values

$$\mathbf{data(x_i) = model(x_i) + error(x_i),\ i = 1, N.}$$

The measured data set has noise, or measurement errors. Usually, the measured data will look something like the model. This is because small values for $error(x_i)$ are usually more likely than large errors. In any case, we would all agree that some of the possible data sets are more probable than others.

If we knew the values of the M parameters which describe the model, and if we knew the probability distribution function for the errors, we could immediately write the probability for obtaining our measured data set. We would know if it were a probable or an improbable occurrence. Clearly, we don't know the values of these M parameters, or we wouldn't be using RazorFit. So we proceed as follows:

We assume that the probability distributions $P(error(x_i))$ for errors are known. We then write a general probability of obtaining any given data set. For example, suppose that the probability distributions for the errors were Normal distributions, such as

$$\mathbf{P(error(x_i)) = \frac{1}{\sqrt{(2\pi)}\sigma_i} exp[-\frac{(error(x_i))^2}{2\sigma_i^2}]}.$$

The σ_i, or the RMS error measured at the x-value x_i, is allowed to be different at each position x_i. However, in assuming that we know the probability distributions, we are stating that we know all the values σ_i.

When the errors at positions x_i and x_j are uncorrelated, then the probability, of obtaining the data set $\{data(x_1), ...data(x_n)\}$ is

$$\mathbf{P(data(x_1), ..., data(x_n)) = \prod_{i=1}^{N} \frac{1}{\sqrt{(2\pi)}\sigma_i} exp[-\frac{(error(x_i))^2}{2\sigma_i^2}]},$$

$$\mathbf{= \prod_{i=1}^{N} \frac{1}{\sqrt{(2\pi)}\sigma_i} exp[-\frac{(data(x_i) - model(x_i))^2}{2\sigma_i^2}]}.$$

The Maximum Likelihood philosophy is that the given data set is very ordinary, and thus representative of all data sets. Of all possible data sets, this one isn't unusual. Maximum Likelihood says that we should find values for the model parameters which maximize the probability $P(data)$.

Maximize the probability as follows: First make the statement that since probabilities are always ≥ 0, maximizing lnP is the same as maximizing P.

$$\mathbf{lnP(data(x_1), ...) = maximum}$$

$$\mathbf{= \sum_{i=1}^{N} ln(\frac{1}{\sqrt{(2\pi)}\sigma_i}) - \sum_{i=1}^{N} \frac{(data(x_i) - model(x_i))^2}{2\sigma_i^2}}.$$

The first sum in the above equation has a constant value, and so can be ignored in the maximization. Therefore, the equation to be solved is

$$\sum_{i=1}^{N} \frac{(\mathrm{data}(x_i) - \mathrm{model}(x_i))^2}{2\sigma_i^2} = \mathrm{minimum}.$$

We have just derived one of the important results of Maximum Likelihood. **When the errors come from Normal distributions, and when the error associated with one data point is uncorrelated with the error at any other data point, then the Maximum Likelihood prescription is the same as the least-squares prescription.**

RazorFit finds the best parameters for the model you specify, using the least-square criterion derived from the Maximum Likelihood principle for data with Normally distributed errors.

Unless you take the trouble to input a noise variance vector, RazorFit makes the simplifying assumption that σ_i is the same for all data points. RazorFit calculates a mean value σ^2, the variance of the noise, according to

$$\sigma^2 = \mathrm{RMS}^2 = \sum_{i=1}^{N} (\mathrm{data}(x_i) - \mathrm{smoothdata}(x_i))^2/N.$$

where $smoothdata(x_i)$ is the result of the Maximum Likelihood smoothing procedure, **ESmooth**, performed within RazorFit. (RazorFit uses the width of your narrowest peak as the smoothing width.) Thus RazorFit uses ${\sigma_i}^2 = \sigma^2$ for all i.

7.9 Downhill to a minimum

RazorFit uses the Levenberg-Marquardt method[1] to find the minimum value of Chisquare. Within RazorFit, Chisquare is calculated as

$$\mathrm{Reduced\_Chisquare} = \frac{\sum_{i=1}^{N}((\mathrm{data}(x_i) - \mathrm{model}(x_i))/\mathrm{RMS})^2}{(N - M)}$$

N is the number of data points, and M is the number of model parameters which are allowed to vary. RMS is the noise obtained from your noise variance vector, or is calculated by RazorFit according to whether you specify normal or Poisson noise.

This statistic, the **Reduced chisquare**, is the more familiar chisquare, divided by the number of degrees of freedom, $N - M$. When the measurement errors come from Normal distribution, and when the model is correct, the reduced chisquare statistic will come from a distribution which has a mean value of 1, and a width of $\sqrt{2/(N - M)}$.

[1] An excellent reference on modeling data, and on the Levenberg-Marquardt method of fitting nonlinear models, is the book *Numerical Recipes, The Art of Scientific Computing*, by W. H. Press, B. P. Flannery, S.A. Teukolsky, and W. T. Vetterling, Cambridge University Press, 1988.

Chisquare can be thought of as a surface in M-dimensional space, where M is the number of variables which are being determined. When you begin the program, and whenever you fix or unfix any of the parameters of your model, RazorFit adjusts the dimensions of the space it is working in. (Some peakfitting programs don't readjust the dimensions of the space when you fix parameters.) Fit knows its current position in that space, from the current values of the M parameters which are allowed to vary, and so can calculate the height of the reduced chisquare surface at that point. All RazorFit does is this: sit at the position given by the initial parameter settings, look around, find the downhill direction, and ooze down into the nearest minimum.

While RazorFit finds a minimum, this minimum is not necessarily the global minimum. If you have chosen the correct model, and if your measurement errors are random, with zero mean, then you should expect that the global minimum of the Reduced Chisquare surface will have a value

$$\mathbf{Reduced\_Chisquare} = 1 \pm \sqrt{(\frac{2}{N - M})}.$$

When RazorFit finds a minimum, it calculates the Reduced Chisquare at that position. If the value is close to 1, it reports **Solution has converged**. If the Chisquare associated with the found minimum is ≥ 2, it states **Solution has found minimum.** We can think of several reasons the Chisquare value associated with the minimum may be too large:

- The model may not be appropriate.
- Your starting parameters may have led RazorFit into the wrong minimum.
- Your measurement errors may not be random, with zero mean.
- Your measurement errors may come from some distribution other than a Normal or a Poisson distribution.
- Possibly your measurement errors are a lot larger in one region of your data set than in other regions.

If RazorFit tells you it has found a minimum, but Reduced Chisquare is $\gg 1$, we suggest you check this list, and decide why the reduced chisquare value was so large. If either of the first two reasons is the fault, you need to make changes in the starting conditions. If the fault lies with the properties of your errors, remember this: RazorFit is a very good least-square fitting procedure. At the desired minimum, a reduced chisquare value of 1 is expected *only* if the measurement errors are Normally- distributed. For other noise distributions, the least-square fit will give you good parameter values, but the confidence limits will not be correct.

7.10 Confidence Limits

RazorFit reports the final values of the M parameters you are fitting, and also gives you confidence limits for the parameter values. How are these confidence limits computed, and under what conditions should you regard them as the truth?

The confidence limits reported by RazorFit are calculated by taking the square root of the diagonal elements of the covariance matrix. The covariance matrix is the inverse of the Hessian matrix, which has components

$$\mathbf{Hessian_{kl}} = \sum_{i=1}^{N} \frac{1}{{\sigma_i}^2} \frac{\partial \mathbf{model(x_i)}}{\partial \mathbf{a_j}} \frac{\partial \mathbf{model(x_i)}}{\partial \mathbf{a_k}}.$$

The a_j, a_k are the model parameters which are being estimated.

The confidence limits reported by RazorFit are the actual standard errors of the parameter estimation if your measurement errors are Normally- distributed, and if a linearization of the model equations are a pretty good estimate of the true model in a small region around the minimum. For other error distributions, the reported confidence limits are not the true standard errors, but may be reported as 'square root of the diagonal elements of the formal covariance matrix'.

7.11 Limitations of RazorFit

We hope our discussion in this chapter has helped you to think about the assumptions you are making when you decide to fit a model to your data. You need to assess the appropriateness of the model, and carefully choose your initial values. If your measurement errors are random, independent, with zero mean, and if the RMS error is constant across your data set, you will get good values for your model parameters.

RazorFit makes no compromises. At times your RMS noise fluctuations are independent of position, *i.e.* that the mean noise is the same on the left end of the screen as on the right. In other cases, this assumption is not correct. If you have such a case, you should generate a noise vector, containing the variances appropriate to each data point, and use this vector to fill VNOISE. (See Chapter 10.1).

The modeling algorithm used by RazorFit, the Levenberg-Marquardt algorithm, is used by nearly every nonlinear peakfitting program. The limitations of the RazorFit (Levenberg-Marquardt) algorithm are shared by most peakfitting programs you will ever encounter. The algorithm is tailored for data sets with random, Normally- distributed noise or measurement errors. Most measurement noise comes from a statistical distribution which is not Normal. However, even when your measurement errors are not Normally-distributed, if they look somewhat like Figure 7.7.1, the RazorFit algorithm is still a good one to use.

Chapter 8

Peakshape Catalog

A catalog of peakshapes follows. For each peak type, the analytical expression used to synthesize the peakshape is given, in terms of the peak parameters which are optimized by **rzrfit**. The formulas used to calculate the total areas are also given. Total peak areas are calculated for the entire interval $(-\infty,+\infty)$.

8.1 Captured DataPeak

Type 0 — Captured DataPeaks are *real data* peakshapes which have been captured out of your data, or from a calibration run, or maybe from some other data file. Choose a peak whose general shape matches the shapes of peaks in the data you wish to fit. Put the Captured DataPeak in the **datapeak** array of **rzrfit**.

rzrfit will then create a DataPeak Family, with peaks that are wider, narrower, taller, shorter, etc. In other words, **rzrfit** will vary the center position **C**, height **H**, width **W**, and asymmetry parameter **A**, while maintaining the same general shape as your Captured DataPeak.

You are in control of the asymmetry of your Family of DataPeaks. You may let **rzrfit** change the asymmetry parameter for a best fit, or require that all your DataPeaks keep the initial asymmetry.

Whenever you select **DataPeak** as the shape for one of your peaks, **rzrfit** will look to the Family of DataPeaks to give you a best-fit in exactly the same way it would use the Gaussian family if you had selected the Gaussian standard analytic shape.

rzrfit uses the industry-standard Levenberg-Marquardt method for fitting DataPeaks, exactly as it uses that method for fitting Gaussians, Lorentzians, etc. The only difference is: **Now you can use real data shapes. You are not limited to analytic peak shapes!**

The **DataPeak** parameters C, H, W, and A are optimized by **rzrfit**.

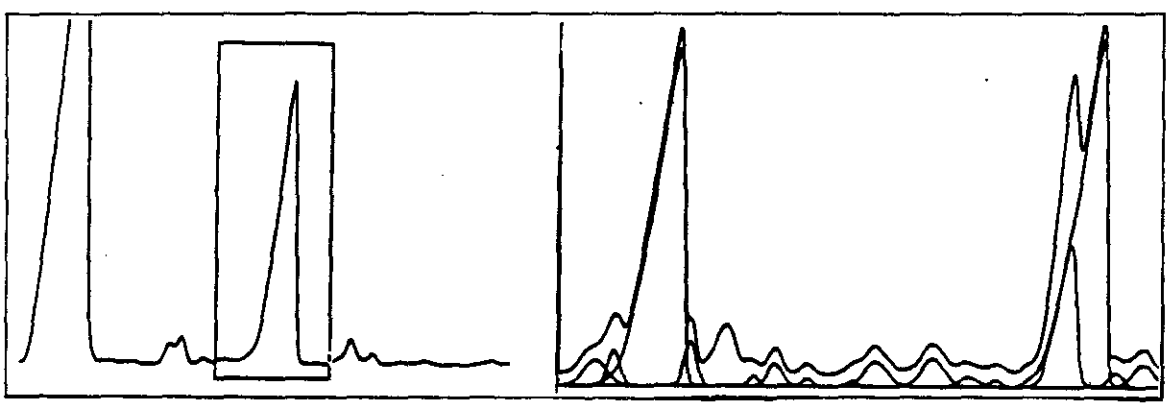


Fig 8.1 **Capture a *real data peak* from a GC run.**
Use **Captured DataPeak** to deconvolve peak overlaps.
rzrfit reports peak centers (elution times), heights, widths, areas, ± standard errors.

8.2 Gaussian

Type 1 — Gaussian peakshapes are synthesized from the equation

$$\mathrm{peak(x)} = \mathrm{Hexp}[-\frac{(\mathrm{x}-\mathrm{C})^2}{(\mathrm{W}/1.665)^2}],$$

where C is the center position of the peak, H is the peak height, and W is the full-width at half-maximum (fwhm). The three parameters C, H, and W are optimized by **rzrfit**. Total areas are $\sqrt{\pi}WH/1.665$.

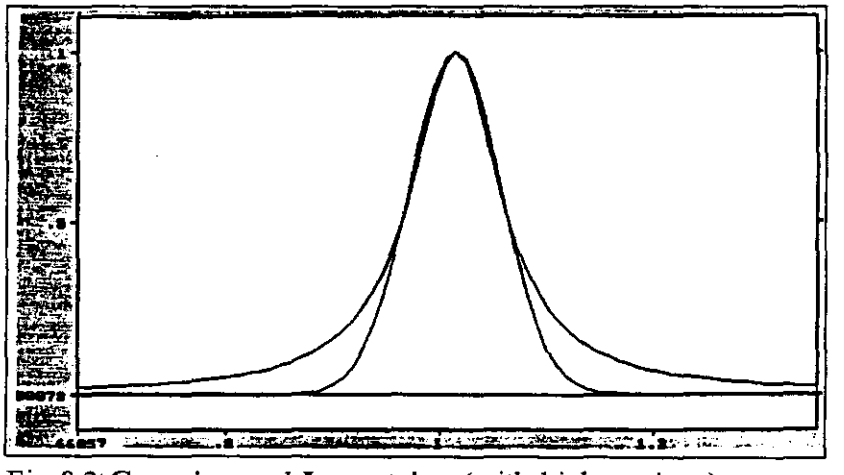


Fig 8.2 **Gaussian** and **Lorentzian** (with higher wings).

8.3 Lorentzian

Type 2 — Lorentzian peakshapes are synthesized from the equation

$$\mathrm{peak(x)} = \mathrm{H}\frac{1}{1+4(\mathrm{x}-\mathrm{C})^2/\mathrm{W}^2},$$

where C is the center position of the peak, H is the peak height, and W is the full-width at half-maximum (fwhm). The three parameters C, H, and W are optimized by **rzrfit**. Total areas are $\pi WH/2$.

8.4 Sum Gaussian + Lorentzian

Type 3 — Sum Gaussian+Lorentzian peakshapes are synthesized from the equation

$$\mathbf{peak(x) = H(1-A)(exp[-\frac{(x-C)^2}{(W/1.665)^2}]) + HA(\frac{1}{1+4(x-C)^2/W^2}),}$$

where C is the center position of the peak, H is the peak height, and W is the full-width at half-maximum (fwhm), and A is the Lorentz fraction (A = 1 for 100% Lorentzian). The four parameters C, H, W, and A are optimized by **rzrfit**. Total areas are $WH(\sqrt{\pi}(1-A)/1.665 + \pi A/2)$.

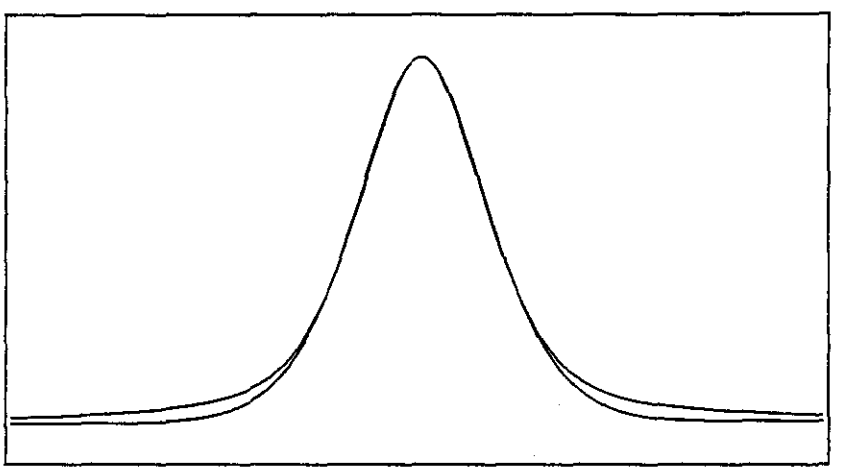


Fig 8.3 **Sum(G+L)** (with higher wings) and **Product(G⋆L)**.

8.5 Product Gaussian⋆Lorentzian

Type 4 — Product Gaussian⋆Lorentzian peakshapes are synthesized from the equation

$$\mathbf{peak(x) = H(exp[-\frac{(x-C)^2}{(W/1.665)^2}]) \star (\frac{1}{1+4(x-C)^2/A^2}),}$$

where C is the center position of the peak, H is the peak height, W is the Gaussian width (fwhm) and A is the Lorentzian width (fwhm). The four parameters C, H, W, and A are optimized by **rzrfit**. Total peak areas are calculated by summing the peaks to the edges $E_{-,+}$ of the data set, and approximating each of the unseen wing areas with the formula $HA^2 exp[-1.665E^2_{-,+}/W^2]/4$.

8.6 Asymmetric Gaussian

Type 5 — Asymmetric Gaussian peakshapes are synthesized from the equation

$$\mathrm{peak(x)} = \mathrm{Hexp}[-\frac{(\mathrm{x}-\mathrm{C})^2}{(2\mathrm{W}(1-\mathrm{A})/1.665)^2}, \mathrm{x} \leq \mathrm{C},$$

$$\mathrm{peak(x)} = \mathrm{Hexp}[-\frac{(\mathrm{x}-\mathrm{C})^2}{(2\mathrm{W}(1+\mathrm{A})/1.665)^2}, \mathrm{x} \geq \mathrm{C},$$

where C is the center position of the peak, H is the peak height, W is the full-width at half-maximum (fwhm), and A is the asymmetry. The four parameters C, H, W, and A are optimized by **rzrfit**. Total areas are given by $\sqrt{\pi}(W+A)H/1.665$.

8.7 Asymmetric Lorentzian

Type 6 — Asymmetric Lorentzian peakshapes are synthesized from the equation

$$\mathrm{peak(x)} = \mathrm{H}\frac{1}{1+4(\mathrm{x}-\mathrm{C})^2/(\mathrm{W}(1-\mathrm{A}))^2}, \mathrm{x} \leq \mathrm{C},$$

$$\mathrm{peak(x)} = \mathrm{H}\frac{1}{1+4(\mathrm{x}-\mathrm{C})^2/(\mathrm{W}(1+\mathrm{A}))^2}, \mathrm{x} \geq \mathrm{C},$$

where C is the center position of the peak, H is the peak height, W is the full-width at half-maximum (fwhm), and A is the asymmetry. The four parameters C, H, W, and A are optimized by **rzrfit**. Total areas are given by $\pi H(W+A)/2$.

8.8 Symmetric and Asymmetric Pearson7

Type 7 — Pearson7 (Pearson VII) peakshapes are often used as approximations to the Voigt shape. The Pearson7 shape can look like a Gaussian, a Lorentzian, or anything in between. In addition it can be used to fit supra-Lorentzian shapes (*very wide* wings) and supra-Gaussian shapes (*very narrow* wings).

Razor Library gives you both a symmetric and an asymmetric Pearson7. Symmetric Pearson7 shapes are synthesized from the equation

$$\mathrm{peak(x)} = \mathrm{H}\frac{1}{(1+4(\mathrm{x}-\mathrm{C})^2(2^{1/\mathrm{A}}-1)/\mathrm{W}^2)^{\mathrm{A}}},$$

where C is the center position of the peak, H is the peak height, and W is the full-width at half-maximum (fwhm). A is a parameter which governs the shape. A

can take any value between 0.5 and ∞. When A=1, the Pearson7 peak is a pure Lorentzian shape. When $A \rightarrow \infty$, the Pearson7 becomes Gaussian. In practice, the Pearson7 shape is close to Gaussian for $A \geq 10$.

Asymmetric Pearson7 shapes are synthesized from the equations

$$\mathrm{peak(x)} = \mathrm{H}\frac{1}{(1+4(\mathrm{x}-\mathrm{C})^2(2^{1/(\mathrm{A}(1+\mathrm{P})}-1)/\mathrm{W}^2)^{\mathrm{A}(1+\mathrm{P})}}, \mathrm{x} \leq \mathrm{C},$$

$$\mathrm{peak(x)} = \mathrm{H}\frac{1}{(1+4(\mathrm{x}-\mathrm{C})^2(2^{1/(\mathrm{A}(1-\mathrm{P})}-1)/\mathrm{W}^2)^{\mathrm{A}(1-\mathrm{P})}}, \mathrm{x} \geq \mathrm{C},$$

where C is the center position of the peak, H is the peak height, and W is the full-width at half-maximum (fwhm). $A(1+P)$ governs the shape. A can take any value between 0.5 and ∞. P can take any value between -2000 and 2000.

The parameters C, H, W, A, and (optionally) P, are optimized by FIT.

If a symmetric Pearson7 is desired, set the parameter $P = 0$, and set $fixp = 0$. (See page 103).

8.9 Log Normal

Type 8 — Log Normal peakshapes[1] are synthesized from the equation

$$\mathrm{peak(x)} = \mathrm{Hexp}[-\frac{(\ln(1+\mathrm{E}(\mathrm{x}-\mathrm{C})))^2}{\mathrm{D}}], \mathrm{x} \geq \mathrm{C}-1/\mathrm{E},$$

$$\mathrm{peak(x)} = 0, \mathrm{x} \leq \mathrm{C}-1/\mathrm{E},$$

where C is the center position of the peak, and H is the peak height. E and D are functions of W, the full-width at half-maximum, and A, the asymmetry factor. $D = (lnA)^2/ln2$. $E = (A^2-1)/AW$. The four parameters C, H, W, and A are optimized by FIT. Total areas are $H\sqrt{D\pi}e^{D/4}/E$.

[1] D. E. Metzler, C. M. Harris, R. J. Johnson, D. B. Siano, J. A. Thomson, (1973), *Biochemistry 12*, 5377.

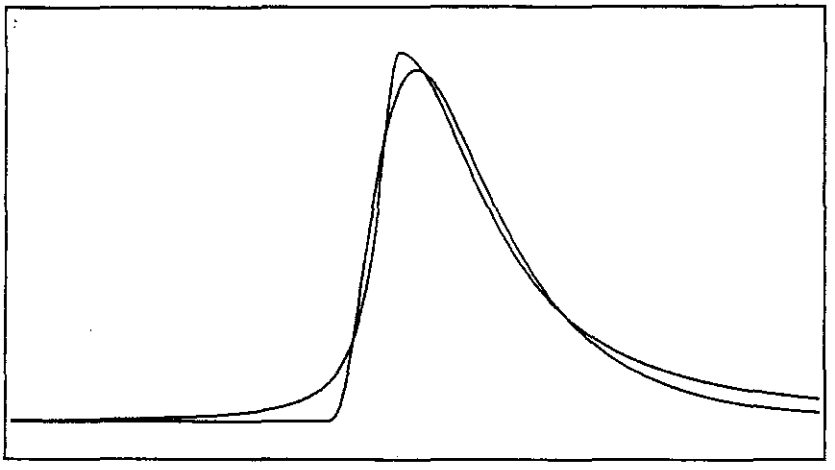


Fig 8.4 **Log Normal** and **AsymLrnz** (with higher wings).

8.10 Baseline types

Type 200 — User baseline, from **rzrbas** or **rzrqba**, or a background scan, saved in a file. User baselines are entered into **rzrfit** through the array **baslin**.

Type 201 — Offset baselines are described by the equation

$$\mathrm{baseline(x)} = \mathrm{B_0}.$$

Type 202 — Linear baselines are described by the equation

$$\mathrm{baseline(x)} = \mathrm{B_0} + \mathrm{B_1 x}.$$

Type 203 — Quadratic baselines are described by the equation

$$\mathrm{baseline(x)} = \mathrm{B_0} + \mathrm{B_1 x} + \mathrm{B_2 x^2}.$$

Type 204 — Exponential baselines are described by the equation

$$\mathrm{baseline(x)} = \mathrm{B_0}[1 + \mathrm{B_1 exp}(-\mathrm{B_2 x})].$$

The constants B_0, B_1, and B_2 are optimized by **rzrfit**.

NOTE: We do not advise using either quadratic or exponential baselines, unless you are sure of your peak types, and also absolutely sure that this is the appropriate baseline to use. Both quadratic and exponential baselines lead to terrible convergence problems when combined with anything other than a few isolated Gaussian peaks.

Chapter 9

Baselines — rzrbas/rzrqba/rzredg/rzrcut

9.1 Baseline Fitting and Removal

Baseline fitting often feels like the black hole of spectral analysis. If the baseline is wrong, peak areas will be wrong — and the errors can be large indeed! This may the hardest problem in the book.

We have no magic bullet. We have tried to apply Maximum Likelihood and Bayesian principles to the problem, but so far we have failed to create a Maximum-anything baseline removal method.

RazorBase is the closest we have come to providing a principled baseline function. It is pretty good! And it is pretty slow. We were thinking of the impatient spectroscopists in the world (only ourselves?) when we went on to create RazorQuickBase. Perhaps we should have looked at our own notice board instead, where we keep the message "If you can't find time to do it right, when will you find time to do it over?"

While we continue to seek the holy grail of baseline functions, please try out our current efforts and let us know the lay of the land.

9.1.1 RazorBase

Our best effort is embodied in **rzrbas**, which works as follows:

- Identify the peaks in the data, using the high-performance Maximum Likelihood/Bayesian peak-picker **rzrpic**.
- Use the results of **rzresm** to give Maximum Entropy- smoothed segments in the off-peak regions.
- Connect the smooth baseline segments with a straight line under the peaks.

- Check that none of the straight line connections intercept the data at a level higher than the measured RMS noise. Use multiple straight line segments if necessary.

rzrbas is presented in Section 9.2, page 139.

9.1.2 RazorQuickBase and RazorEdge

RazorQuickBase (**rzrqba**) uses a 'quickpick' algorithm to estimate the positions of peaks in the data, and then proceeds in the same manner as **rzrbas**. It is not as reliable at finding small peaks, but it brings results an order of magnitude faster. **rzrqba** is presented in Section 9.4, page 147.

RazorEdge (**rzredg**) attempts to match the baseline to the lower edge of the data. It is very fast, and often works on data which do not yield well to other baseline functions. **rzredg** is presented in Section 9.6, page 152.

9.2 rzrbas

rzrbas estimates the baseline for a spectrum by identifying the peaks. The function **rzrpic** is used in its high- performance mode for peak identification. Thus **rzrbas** requires all the same input as **rzrpic**, and more.
Required user input:

- Data set containing peaks.
- Select a peakshape which represents the peaks in the data set. The peakshape choice is not very critical for this algorithm.
- When the selected peakshape is positive, **rzrbas** will search for positive peaks; when the peakshape is negative, negative peaks in the data will be identified.

Processing notes:

- **rzrbas** finds peaks which have the declared peakshape in the data, using the same algorithm used by **rzrpic**. **rzrbas** will search for negative peaks if the peakshape presented in **shape** is a negative peak.
- **rzrbas** assigns significances to the peaks in signal/noise units. Both height/noise and area/noise significances are available. The heights and areas are calculated from the heights and widths estimated using the shape of the 2nd derivative curve. Significances, heights, and widths are returned in the **sigpks** array.

Programming notes:

- Set **istat** = 1 if the noise is Normal. Set **istat** = 2 if the noise is Poisson.
- Set **psens** = 3 to find all the peaks with heights > 3 times the RMS noise, i.e. all peaks with signal/noise ratios > 3. Set **psens** = -3 to find all the peaks with areas > 3 times the RMS area-noise.
- **bsens** controls the sensitivity of the peak vs. baseline allocation. It is not an arbitrary parameter, but is based upon the expected RMS fluctuations at different spatial frequencies in Normal noise. In theory, it should have worked as reliably and effectively as **psens**.

  In practice, we have had to adjust **bsens** for different types of data. In retrospect, we should have known it would happen. Real baselines do not have anything approaching a Normal distribution of spatial frequencies!

  Start with **bsens** = 1, and adjust it until the value is right for your type of data. Larger values of **bsens** move the baseline/peak junctions in toward the peak centers, allocating more points to the baseline segments.

rzrbas will perform the adjustment for a new value of **bsens** very quickly. It does not need to go back through the peak- picker for this adjustment. Try out **rzrbas** in the demonstration program HandleG to see this in action.

- The small arrays **locpks** and **sigpks** return useful information. See the discussion for **rzrpic**.

- The small array **ibase** returns useful information about peak start/stop regions. **ibase[0]** contains the start index of the first peak; **ibase[nibase/2]** contains the stop index of the first peak. And so on for up to **nibase/2** (start,stop) pairs. **ibase** is filled with the value -1 (an illegal index value) where it is not being used for (start,stop) pairs.

long rzrbas(float ydata[ ], long n2, float shape[ ], long nl2, float yout[ ],
float w[ ], float v[ ], float trans[ ],long *n, long *newpk, long *newbas,
long *istat, double *psens, double *bsens, long locpks[ ], long *npks,
float sigpks[ ], long nsig, long ibase[ ], long nibase, long *nfwhm,
double *peak, 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**

w, length **n**

v, length **n**

trans, length **n**

locpks, length **npks**

sigpks, length **nsig**

ibas, length **nibas**

Input variables: **n2, nl2, n, newpk, newbas, istat, npks, nsig, psens, bsens, nibas**

n2 is the last position of data in **ydata**

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

n is the size of arrays **ydata, yout, w, v**, and **trans**

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

newbas is an initialization flag for **rzrbas**.

istat is a flag for Normal vs. Poisson noise.

npks is the size of the **locpks** array.

nsig is the size of the **sigpks** array.

psens is the threshold peak sensitivity in S/N units.

bsens is the threshold baseline sensitivity in S/N units.

nibase is the size of the **ibas** array.

Output arrays:

yout, filled with baseline

w, filled with smoothed data between **0** and **n2**

locpks, filled between 0 and **npks**-1

sigpks, filled between 0 and 3***npks**-1

ibas, filled between 0 and **nibas**-1

Output variables:

n = amount of array space used

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

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

npks = number of peaks detected
nfwhm = full-width-at-half-maximum of peakshape in **shape**
peak = height of peakshape in **shape**
sigma = RMS noise in the **ydata**.

Function return values:

rzrbas = 0 if operation was successful
If **rzrbas** < 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.

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**. If the peakshape is right-side up, positive peaks will be identified by **rzrbas**. If the peakshape is a negative peak, then negative peaks will be found.

nl2 is *input* and the *index of the last data point of the peakshape* in **shape**.

yout is the *output baseline*. The baseline will be found between data points **0** and **n2**, and should be ignored outside this range.

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

w is a *work array* of length at least **n**.

On *output*, **w** contains a *smoothed data file*. The smoothing has been done by **rzrbas**.

v is a *work array* of length at least **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 **rzrbas**. 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**, **v**, and **trans** arrays.

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

n = rzsizn(n2,shape,nl2)

On *output*, **n** is the amount of *space used for the Fourier transforms* in the **yout**, **w**, **v**, 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 **rzrbas** returns after successful processing, it fills both **newpk** and **n** with the transform size. If you with to process additional data with the same peakshape, you need not change either **newpk** or **n**, provided that (a) your peakshapes do not change, and (b) your input **ydata** sizes (**n2**+1) do not increase.

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

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1*. Whenever **rzrbas** 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 **rzrbas** will be **rzrbas** = -2.

istat is an *input flag which governs the statistics* used by the function. Set **istat** = 1 if the noise is Normal. Set **istat** = 2 if the noise is Poisson.

psens is an *input S/N threshold variable* that directs the peak picker. The peak picker assigns each peak a significance in units of the RMS noise. **rzrbas** returns peaks whose significances exceed the value **psens**. When **psens**=0.0, all possible peaks are found. When **psens** = 3.0, all peaks with heights > 3.0 RMS noise (i.e. S/N > 3.0) are returned. When **psens** = -3.0, all peaks with areas > 3.0 RMS area-noise

are returned. Peaks which are at least 3 to 5 times the RMS noise are meaningful (**psens** = 3 to 5).

bsens controls the sensitivity of the peak vs. baseline allocation. Start with **bsens** = 1, and adjust it until the value is right for your type of data. Larger values of **bsens** move the baseline/peak junctions in toward the peak centers, allocating more points to the baseline segments.

rzrbas will perform the adjustment for a new value of **bsens** very quickly. It does not need to go back through the peak- picker for this adjustment. Try out **rzrbas** in the demonstration program HandleG to see this in action.

newbas is an *input initialization flag and also a picker-selection flag.* **newbas** works inside **rzrbas** in exactly the same way as **iperf** works for **rzrpic**. Set **newbas** = 1 to get the High-Performance Bayesian pickker, **newbas** = 2 to get the High Resolution picker, **newbas**= 3 to get the 2nd-Order Asym picker, and **Newbas** = 4 for the Quiet Picker. Set **newbas** = -1 for the Quick Pick.

locpks is an *output integer array containing the peak locations*, i.e., **locpks**(0) = data point number of the first peak detected. **locpks** need be no larger than the maximum number of peaks expected. **locpks** and **npks** may be used as input to **rzrfil**.

npks is *input* as *the size of array* **locpks**.

npks is *output* as the *number of peaks* located by the search. Thus, the array **locpks** will be filled with meaningful numbers between **locpks**(0) and **locpks(npks**-1).

sigpks is an *output array containing the peak significance* assigned by **rzrbas**. The significance is in units of the RMS noise in the data set. The output arrays **locpks** and **sigpks** are sorted by significance. **sigpks**(0) $\geq$ **sigpks**(1), etc.

The length of the **sigpks** array should be 3***npks** = three times the maximum number of peaks expected. This will provide room to report the peak significances, peak heights and peak widths.

The contents of the **sigpks** array will be **sigpks**(0) = significance assigned to the peak found at data point number **locpks**(0), **sigpks**(*npks) = height assigned to the peak found at data point number **locpks**(0), **sigpks**(*npks*2) = width assigned to the peak found at data point number **locpks**(0),.

nsig is *input* as *the size of array* **locpks**.

The minimum length of the **sigpks** array is **nsig** = **npks**. This provides enough room to return peaks significances in **sigpks**.

To obtain peak heights and peak widths in **sigpks**, as well as peak significances, set **nsig** = 3***npks** = three times the maximum number of peaks expected.

ibase is filled on *output* with peak start/stop regions. **ibase[0]** contains the start index of the first peak; **ibase[nibase/2]** contains the stop index of the first peak. And so on for up to **nibase/2** (start,stop) pairs. **ibase** is filled with the value -1 (an illegal index value) where it is not being used for (start,stop) pairs.

nibase is *input* as the size of the array **ibase**. **nibase** must be twice as big as the number of peak *regions* expected. If **ibase** is too small to contain a list of all the (start,stop) peak regions in the data, the final baseline presented in **yout** will be in error.

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

peak is *output* as the *height of the peakshape* in the array **shape**.

sigma is *output* as the *standard deviation (root- mean-square) of the noise* which was found in **ydata**.

9.3 Example using rzrbas

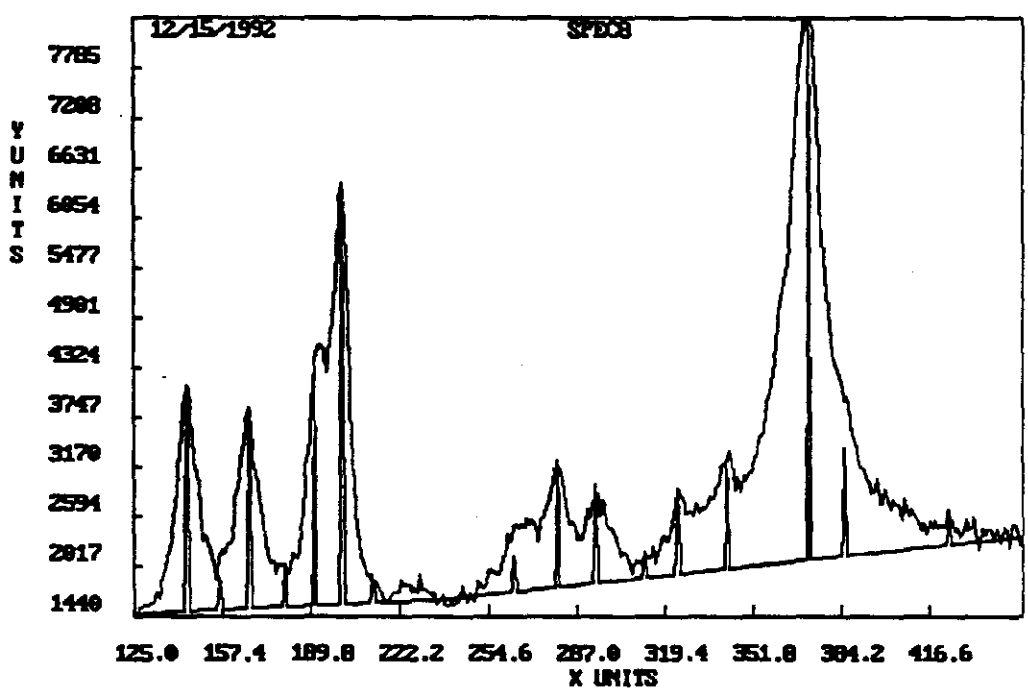
SPEC8 is a Raman spectrum of ethyl acetate. The noise statistics are Poisson.

Spectrum file: SPEC8
Peakshape file: PEAK8

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): BAS

Enter name of spectrum (Try SPEC8): SPEC8
Enter name of peakshape (Try PEAK8): PEAK8

RZRBAS also picks peaks! Choose picker.
Enter -1 for Quick-Pick
Enter 1 for High-Performance picker
Enter 2 for High-Resolution picker
Enter 3 for 2nd-Order High-Performance picker
Enter 4 for Quiet-Pick (High Res. Good for very narrow peaks.)
Select picker by Number [2]: 2

Enter peak threshold (# of standard deviations of noise):
(i.e., Enter 3 for peaks with HEIGHTS ¿ 3 RMS noise)
(Enter -2 for peaks with AREAS ¿ 2 RMS Area-noise)
Enter 3 if unsure: 3
```

```
Enter N for Normal noise; P for Poisson noise [N]: N

Entering RZRBAS with bsens=1.0      Wait for setup...
Estimated RMS noise:        77.96
Using Peak detection threshold:  3.0
Number Peaks detected:          21

Peak Start/Stop regions:
   0/ 115    115/ 324

Current baseline sens, BSENS=1.0
Enter new value for BSENS (Enter 0 to quit): 0

BASELINE MAY BE SAVED TO A FILE AFTER EXAMINING PEAKS.
Hit any key to examine peak parameters.

Peaks found by RZRBAS
Estimated RMS noise:        77.96
Number Peaks detected:          21
Using Peak detection threshold:  3.00000
Peak HIEGHT Significances:
    85.82 58.24 .........

Select:
Print (S)ignificances, (L)ocations, (H)eights, (W)idths, (E)verything.
(R)esort. (T)uneup heights. (A)ccept. (M)enu.  M
```



9.4 rzrqba

rzrqba estimates the baseline for a spectrum by identifying the peaks, smoothing the off-peak segments, and joining the smoothed segments with straight lines under the peaks. The difference between **rzrqba** and **rzrbas** is that **rzrqba** does not use a Maximum Likelihood/Bayesian 2nd derivative in its peak picker, and does not use Maximum Entropy to smooth the off-peak baseline segments.
Required user input:

- Data set containing peaks.
- Select a peakshape which represents the peaks in the data set. The peakshape choice is not very critical for this algorithm.
- When the selected peakshape is positive, **rzrqba** will assume positive peaks; when the peakshape is negative, it will assume negative peaks.

Programming notes:

- **bsens** controls the sensitivity of the peak vs. baseline allocation. It is not an arbitrary parameter, but is based upon the expected RMS fluctuations at different spatial frequencies in Normal noise. In theory, it should have worked as reliably and effectively as **psens**.

  In practice, we have had to adjust **bsens** for different types of data. In retrospect, we should have known it would happen. Real baselines do not have anything approaching a Normal distribution of spatial frequencies!

  Start with **bsens** = 1, and adjust it until the value is right for your type of data. Larger values of **bsens** move the baseline/peak junctions in toward the peak centers, allocating more points to the baseline segments.

  rzrqba will perform the adjustment for a new value of **bsens** very quickly. It does not need to go back through the peak- picker for this adjustment. Try out **rzrqba** in the demonstration program HandleG to see this in action.

- The array **ibase** returns useful information about peak start/stop regions. **ibase[0]** contains the start index of the first peak; **ibase[nibase/2]** contains the stop index of the first peak. And so on for up to **nibase/2** (start,stop) pairs. **ibase** is filled with the value -1 (an illegal index value) where it is not being used for (start,stop) pairs.

long rzrqba(float ydata[ ], long n2, float shape[ ], long nl2,
float yout[ ], float w[ ], float v[ ], long *newbas,
double *bsens, long ibase[ ], long nibase, long *nfwhm, double *peak, double *sigma)

Input arrays which must be filled:
ydata, filled between **0** and **n2**, length **n2 + 1**
NOTE: **ydata** will be read only, not altered.
shape, filled between **0** and **nl2**
NOTE: **shape** will be read only, not altered.
Additional arrays to be furnished:
yout, length **n**
w, length **n**
v, length **n**
ibas, length **nibas**
Input variables: **n2, nl2, newbas, bsens, nibas**
n2 is the last position of data in **ydata**
n2+1 is the size of arrays **ydata, yout, w**, and **v**
nl2 is the last position of data in **shape**
newbas is an initialization flag for **rzrqba.**
bsens is the threshold baseline sensitivity in S/N units.
nibase is the size of the **ibas** array.
Output arrays:
yout, filled with baseline
ibas, filled between 0 and **nibas**-1
Output variables:
nfwhm = full-width-at-half-maximum of peakshape in **shape**
peak = height of peakshape in **shape**
sigma = RMS noise in the **ydata**.
Function return values:
rzrqba = 0 if operation was successful
If **rzrqba** < 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.

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**. If the peakshape is right-side up, positive peaks will be assumed by **rzrqba**. If the peakshape is a negative peak, then negative peaks will be assumed.

nl2 is *input* and the *index of the last data point of the peakshape* in **shape**.

yout is the *output baseline*. The baseline will be found between data points **0** and **n2**. **yout** must have a size equal to (**n2**+1).

w is a *work array* of length = **n2**+1.

v is a *work array* of length = **n2**+1.

newbas is an *input initialization flag*. It should be set = 1 for the initial call. **rzrqba** will maintain **newbas** after that.

bsens controls the sensitivity of the peak vs. baseline allocation. Start with **bsens** = 1, and adjust it until the value is right for your type of data. Larger values of **bsens** move the baseline/peak junctions in toward the peak centers, allocating more points to the baseline segments.

rzrqba will perform the adjustment for a new value of **bsens** very quickly. It does not need to go back through the peak- picker for this adjustment. Try out **rzrqba** in the demonstration program HandleG to see this in action.

ibase is filled on *output* with peak start/stop regions. **ibase[0]** contains the start index of the first peak; **ibase[nibase/2]** contains the stop index of the first peak. And so on for up to **nibase/2** (start,stop) pairs. **ibase** is filled with the value -1 (an illegal index value) where it is not being used for (start,stop) pairs.

nibase is *input* as the size of the array **ibase**. **nibase** must be twice as big as the number of peak *regions* expected. If **ibase** is too small to contain a list of all the (start,stop) peak regions in the data, the final baseline presented in **yout** will be in error.

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

peak is *output* as the *height of the peakshape* in the array **shape**.

sigma is *output* as the *standard deviation (root- mean-square) of the noise* which was found in **ydata**.

9.5 Example using rzrqba

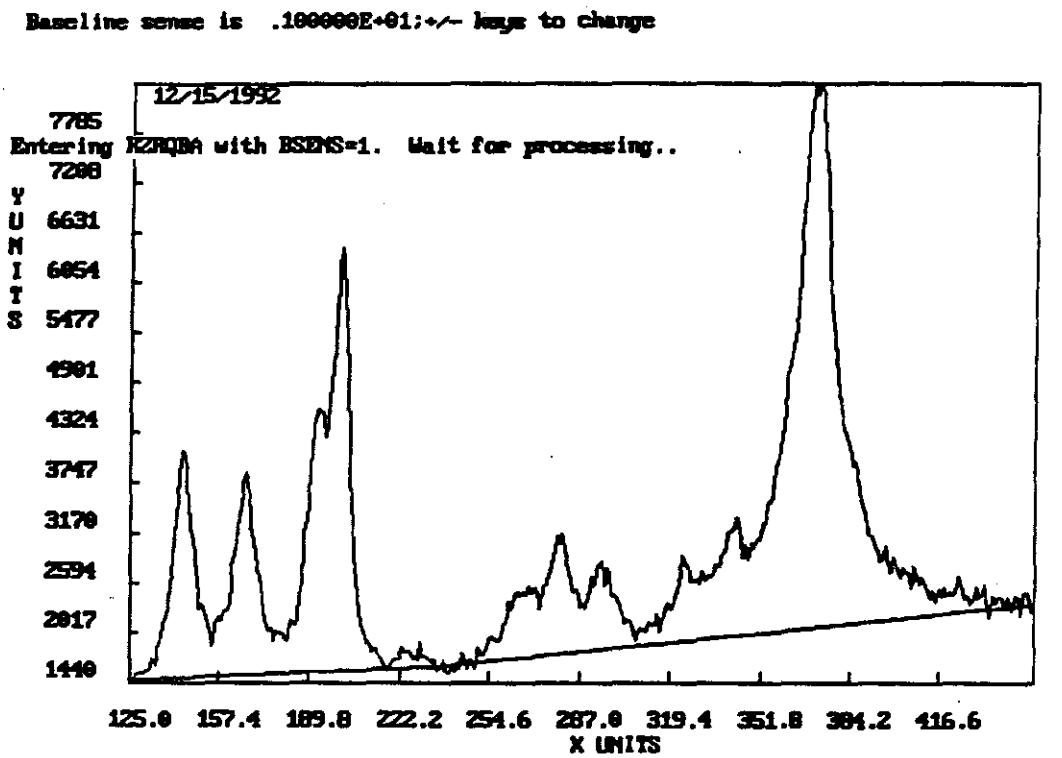
SPEC8 is a Raman spectrum of ethyl acetate. The noise statistics are Poisson.

Spectrum file: SPEC8

Peakshape file: PEAK8

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): QBA

Enter name of spectrum: SPEC8
Enter name of peakshape: PEAK8

Enter RZRQBA with bsens = 1. Wait for setup..
The FWHM of the peakshape is 9
The RMS noise in the data is 122.992
Peak Start/stop regions:
  5/112  112/281  282/286  286/305  308/309
Current baseline sens, BSENS = 1
Enter new value for BSENS (Enter 0 to quit):

Press ENTER to return to menu.
```


Baseline sense is .100000E+01; +/- keys to change
12/15/1992
Entering RZRQBA with BSENS=1. Wait for processing..
7785
7208
6631
6054
5477
4901
4324
3747
3170
2594
2017
1440
Y UNITS
125.0 157.4 189.8 222.2 254.6 287.0 319.4 351.8 384.2 416.6
X UNITS

9.6 rzredg

rzredg tries to find a baseline that cuts underneath all the data, but does not push up into the peaks.
Required user input:

- Data set containing peaks.
- Select a peakshape which is *at least as wide as the widest peaks in the data set.* The peakshape choice is not very critical for this algorithm.

Programming notes:

- **nbsens** controls the sensitivity of the baseline allocation.

  In practice, we have found it conveneint to be able to adjust **nbsens** for different types of data. Start with **nbsens** = 5, and adjust it until the value is right for your type of data. Larger values of **nbsens** move the baseline upward and into the peaks.

long rzredg(float y[ ], long n2, float shape[ ], long nl2, float x[ ], float w[ ], float z[ ], long bnsens, long *nfwhm

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

Additional arrays to be furnished:
- **yout**, length **n2**
- **w**, length **n2**
- **v**, length **n2**

Input variables: **n2, nl2, nbsens**
- **n2** is the last position of data in **ydata**
- **n2** is the size of arrays **ydata, yout, w**, and **v**
- **nl2** is the last position of data in **shape**
- **nbsens** is the baseline sensitivity

Output arrays:
- **yout**, filled with baseline

Output variables:
- **nfwhm** = full-width-at-half-maximum of peakshape in **shape**

Function return values:
- **rzredg** = 0 if operation was successful
  - If **rzredg** < 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 **1** and **n2**. **ydata** will NOT be altered outside this range.

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 **1** and **nl2** in **shape**.

nl2 is *input* and the *index of the last data point of the peakshape* in **shape**.

yout is the *output baseline*. The baseline will be found between data points **1** and **n2**. **yout** must have a size equal to (**n2**).

w is a *work array* of length = **n2**.

v is a *work array* of length = **n2**.

nbsens controls the sensitivity of the baseline allocation. Start with **nbsens** = 5, and adjust it until the value is right for your type of data. Larger values of **nbsens** move the baseline up and into the peak centers.

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

9.7 rzrcut

rzrcut produces a cubic spline fit to spectral point pairs selected by the user; it is one of the better ways to fit a smooth curve underneath spectra, since it is quite free from the inappropriate oscillations which characterize polynomial baseline algorithms.

Required user input:

- A list of x,y data pairs through which the spline curve will pass.

Programming notes:

- The input arrays **a** and **b**, which contain the user's x,y pairs, must be sorted in ascending order of x-values.

long rzrcut(float yout[ ], long n2,
float a[ ], float b[ ], float c[ ], float d[ ], long npts,
double derl, double derr)

Required input arrays:
a = an array of chosen abcissa points, length **npts**
b = the corresponding ordinate array, length **npts**
Additional arrays to be furnished:
yout, length $\geq$ **n2**+1
c, length **npts**
d, length **npts**
Input variables: **n2, npts, derl, derr**
n2 is the last position of data in **yout**.
npts is the length of the **a, b, c, d** arrays.
derl, derr = derivatives of input data, at the left and right endpoints, = derivatives at **0** and **n2**.
NOTE: If **derl, derr** $\geq$ .99E30, then the 'natural' spline, with the 2nd deriv=0 at both boundaries is used.
(If you are unsure about what to do, use **derl, derr**=1.0E30).
Output arrays:
yout, filled between **0** and **n2**
Function return values:
rzrcut = 0, if operation was successful
If **rzrcut** < 0, error occurred
Use **rzrerr** (page 174) to obtain error text

Description of variables

yout is an *output* array, containing *the spline fit data*, between **0** and **n2**.

n2 is an *input* value, indicating the last *location to be filled* in the **yout** array.

a, b are *input* data arrays of size **npts**, containing the selected *abcissa and ordinate point pair values* through which the fit must pass.

c, d are *work arrays* of the same size as **a** and **b**.

npts is *input*, the size of the arrays **a, b, c**, and **d**.

derl, derr are *input* values. They are the *derivatives of the spline fit* at the left and right boundaries (**a(0),a(npts-1)**). If **derl, derr** $\geq$ 1.E30 then the 'natural' derivative at these points is assumed.

Chapter 10

RazorNoise — rzrnoi

Noise Estimation

RazorNoise estimates the noise vector of a data file, using a Maximum Likelihood smoothing method. The output can be useful in finding a noise variance vector (**vnoise**) for an input to RazorFit. If the noise is larger at the ends of a file, or if it is bigger at the positions of the peaks, one should use that knowledge in the type of model fitting done by RazorFit. In order to get **vnoise** from the RazorNoise output, one should (a) average the values over nnn adjacent data points, where nnn is large enough to obtain a meaningful average (nnn = 10***nfwhm** is a good place to start), and then (b) square the result.

10.1 rzrnoi

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

- Data array.
- Peakshapes - either true or estimated. It is not critical that the user choose an exact peakshape for **rzrnoi**. 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.

Programmer notes:

- **rzrnoi** requires 3 full-sized arrays, **ydata, yout**, and **trans**.
- **ydata** *will not* be altered by **rzprep** outside the data region **0 - n2**.

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

Input arrays which must be filled:
ydata, filled between **0** and **n2**, length **n2+1**
shape, filled between **n0** and **nl2**
NOTE: **shape** will be read only, not altered.
NOTE: If **newpk** > 1, **shape** will not be read.
Additional arrays to be furnished:
yout, length **n**
trans, length **n**
Input variables: **n2, nl2, n, newpk**
n2 is the index of the last data value in **ydata**
nl2 is the index of the last data value in **shape**
n is the size of arrays **ydata, yout** and **trans**
newpk indicates whether or not **shape** is a new peakshape
Output arrays:
yout, filled between **0** and **n2**
Output variables:
n = amount of array space used
NOTE: if **n** is returned negative, abs(n) = amount of array space needed (but not available). Operation not successful.
newpk = n if **trans** was successfully loaded
nfwhm = full-width-at-half-maximum of peakshape
sigma = RMS noise in **ydata**
Function return values:
rzrnoi = 0 if successful
If **rzrnoi** < 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**. The minimum size of **shape** is **nl2**+1. **nl2** must always be less than **n**.

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

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

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

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

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

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

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

n = rzsizn(n2,shape,nl2)

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

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

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

newpk on *input* is an *integer flag* set which should be initially set to 1. It informs the peakshape processor that a new peakshape is present in **shape**. The processor measures certain parameters of the new peakshape, and then fills the **trans** array

with the Fourier transform of a properly shifted and scaled peakshape. When the peakshape processor finishes successfully, it will *output* **newpk** = **n**, where **n** is the actual space used in **trans**.

The peakshape processor uses that valuable commodity, CPU time, for a Fourier transform. On *input*, the programmer can *circumvent the peakshape processor with* **newpk** > *1*. Whenever **rzrnoi** 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 **rzrnoi** will be **rzrnoi** = -2.

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

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

10.2 Example using rzrnoi

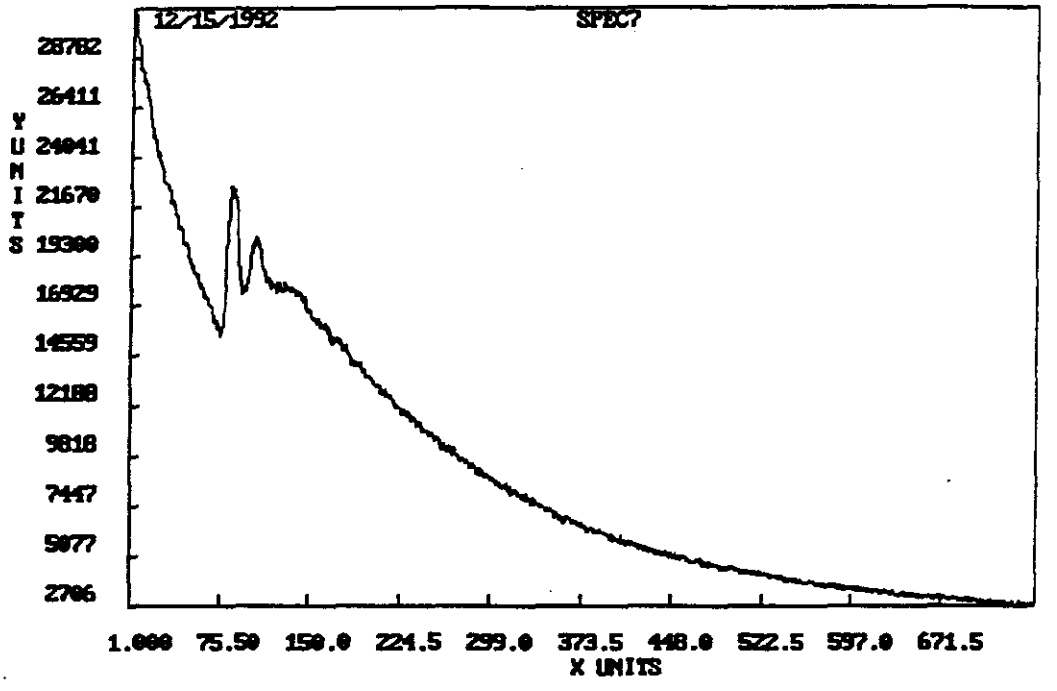
SPEC7 is an EELS spectrum of magnesium oxide. The noise appears to be Poisson, judging from the noise vector produced by **rzrnoi**.

Data file: SPEC7

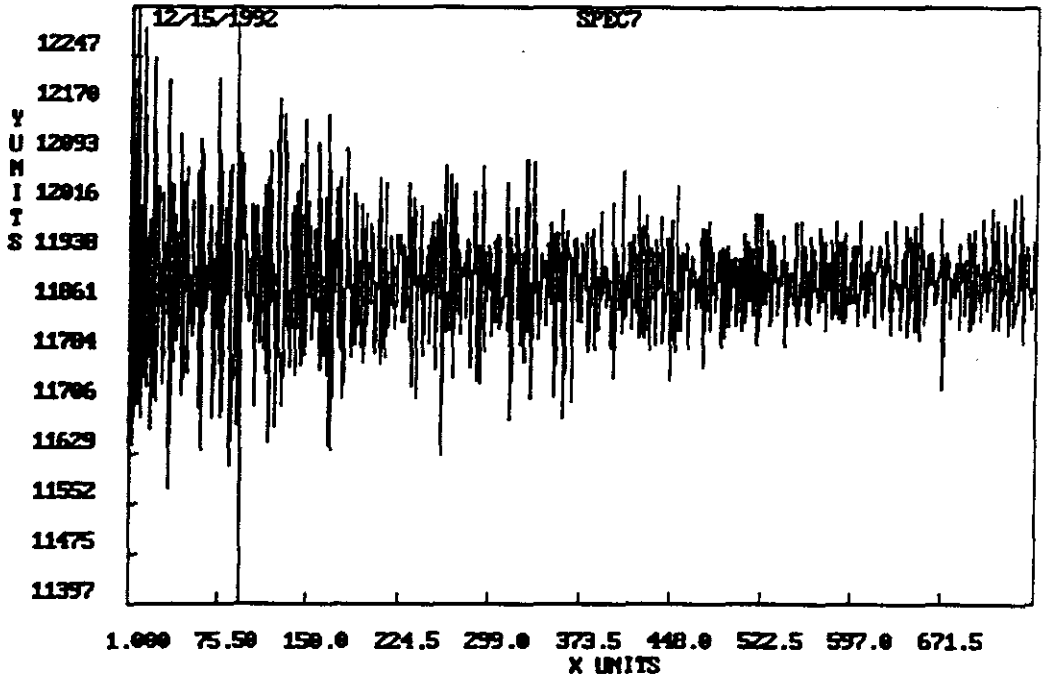
Peakshape file: PEAK7

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): NOI
Enter name of spectrum: SPEC7
Enter name of peakshape: PEAK7
Entering RZRNOI. Please wait for porcessing...
The noise array has been computed and may be saved.
Mean values of the noise variance and standard deviations are:
Variance=9220.1  Standard Deviation= 96.02
```




```
NOI = RazorNoise: Mean noise variance and standard deviation are:
Variance= .918677E+04;  Standard Deviation= .958476E+02
```



Chapter 11

Service Functions

We have provided source code for many functions, including those which we think you may wish to modify. The most important ones deal with time and headaches. You will save a lot of time if you equip yourself with the fastest Fourier transform you can find. You will avoid headaches if you use (a) **rzsizn** (Page 172) to get the correct array sizes, (b) **rzrxpk** (Page 179) to remove a baseline from, and optionally smooth, your **datapeak** or peak **shape**, and (c) **rzrerr** (Page 174) to find out what went wrong.

Many additional little service functions are given in source code in **rzrserve**, so that you may use them if the need arises. **All of the functions in rzrserve are used by the principal functions of Razor, so do not modify the calls or functions, except for changes as discussed below.**

11.1 Fourier transforms — for speed

Most of the Razor principal functions require a Fast Fourier Transform (FFT). The function **rzrft** provides this service for the Razor Library.

All of the principal functions except **rzrqba**, **rzredg** and **rzrcut** call **rzrft(float *a,long n,long iflag)** where **a** is the real **n**-point array to be transformed, and **iflag** = +1 for a forward transform and -1 for an inverse transform. The transform is for real data, and is packed; that is, the last real point is assumed to be found in the imaginary space of the first word. Normalization of the FFT must be carried out during the inverse, not forward, transformation.

You may substitute **rzrft** with any other FFT function that meets the above criteria for syntax and normalization. We recommend that you obtain the *fastest* FFT you can find. We give an example of an alternate **rzrft** below. It is the one you will want if you are using the Microway FFT programmed by Doug Rife.

11.2 Transform padding — rzprep

FFTs require that one pay attention to details such as padding out data arrrays until they occcupy a designated size. The Razor Library assumes power-of-two transforms. **rzprep** is the function used for calculating the size of the array which will be used, and for filling it. We have had *many* discussions among ourselves about which filling method to use. At various times in the past 30 years, we have used straight-line fills, cosine fills, and flip-fills. At present, we agree (pretty well anyway) that a spline-fill is a fair compromise. The spline-fill uses a minimum amount of space, while avoiding infinite derivatives.

We recognize that some users may wish to use different padding and fill. One alternate is the flip-fill method, which we have already coded for you. Flip- fill uses array sizes which are the next power-of-2 greater than twice the data array. The data set is sliced in half, folded outward past the two end points, and then duplicated. The two ends of the folded-out data set are joined together with a flat line. If you wish to use flip-fill, just follow the directions below for replacing the spline fill with flip-fill.

```
/* ************************************************************************
* rzprep(y,nfwhm,n1,n2,n,frac,newpk)
*
* rzprep is used by rzresm, rzrpsm, rzrpic, rzrash, rzrdiv, & rzrnoi.
* It prepares the data in array y for the fft.
*
* summary of input :
* nfwhm is width of peakshape
* n1 = starting position of data in array y
* n2 = last point of data in array y
* n = maximum available array sizes (for y, trans, etc.)
* frac = not used for input
* newpk=1 if this is a new problem (reload trans array from shape)
* newpk=size of trans if this is a continuing problem (trans is loaded)
*
* first rzprep calculates a size n for the fft (size based on data n1,n2)
* suggested ways to calulate n
* n=n2-n1+1+nfwhm*3 (minimum!)
* n=(n2-n1+1)*2  (maximum!)
*
* then rzprep rounds n up to the next power-of-2
*
* Case A. when new power-of-2 larger than array sizes (=input value of n),
* sets n = - power-of-2 calculated for fft
* no action taken on y
* return( -1)
```

```
*
* Case B. when new power-of-2 is ¡= array sizes:
*
*    if input newpk=1,
* prepares y for fft in an array of size n
* using either a flip-fill or a cos-fill method
* flip-fill uses more space, but is more robust for
* very noisy data sets
* cos-fill uses less space, and is implemented here
*
* (if you wish to use flip-fill, comment out the cos-fill code
* (in the subroutine, and un-comment flip-fill code.)
*
* output n = power-of-2 used for fft
* return(0)
*
*    if input newpk¿1,
* this is a sign that trans is already loaded.
* rzprep assumes that current value of newpk is size of trans,
* and rzprep will attempt to load y
* for a transform with that same size.
* however, if new data needs bigger transform size,
* no action taken on y
* return( -2)
* if old transform size is satisfactory,
* y is prepared for fft
* return(0)
*
*    if input newpk=0,
* This value for newpk is no longer used to indicate that the
* shape and trans arrays are unchanged.
* Instead of entering newpk=0, leave newpk alone to
* indicate trans should not be reloaded from shape.
*     Else, set newpk=1 for reloading.
*
* Summary of output:
* n=size used for fft
* (n is negative if arrays not big enough)
* frac=fraction of prepared array y which contains noise
* newpk=unchanged by rzprep if y is successfully prepared
*
* rzprep = 0 when y is successfully prepared for fft
```

```
* = -1 when array sizes (input n) not big enough for desired fft.
* (You need to do 1 thing: provide bigger arrays.
*  Look at -n returned.  Allocate arrays this big,
*  then change -n to n.)
* = -2 when desired fft will be bigger than current size of trans.
* (You need to do 2 things:
*  1. Set newpk=1, to reload trans from shape.
*  2. Look at -n returned.  Allocate arrays this size,
*    then change -n to n.)
* = -3 when incoming newpk=0.  This flag is no longer used.
*
* NOTE: When rzprep¡¿0, the value is passed directly back to the
* programmer as the return value of the routines
* rzresm,rzrpsm,rzrpic,rzrash,rzrdiv,& rzrnoi.
*
*************************************************************************
/* rzprep uses Razor Library functions
rzpowr
*/
#pragma comment(exestr," (c) Copyright 1991-96 Spectrum Square Associates, Inc '
long rzprep(float y[], long nfwhm, long n1, long n2, long *n,
double *frac, long *newpk)
-
   long i, nhold;
   long m;
   double arg;
   long nwidth;
   double slout, slin;
   float z[3];
   float a[3];
   float b[3];
   float u[3];
   *frac = 1.0;
   nhold = *n;

/* BEGIN code for cos-fill and for spline-fill PART 1/2
* This is the code for a cos-fill.  If you use this fill method,
* replace part 1 of the flip-fill code in this function
* with the code shown here:
*/
   *n = n2 -n1 +1 +nfwhm*3;
/* END code for cos-fill and spline-fill PART 1/2
```

```
*/

/* BEGIN code for flip-fill  PART 1/2
* This is the code for a flip-fill.  If you use this fill method,
* replace part 1 of the cos-fill code in this function
* with the code shown here:
*/
/*
   *n = (n2 - n1 + 1)*2;
*/
/* END code for flip-fill PART 1/2
*/

   if( *n ¡= n2 )
 *n = n2 + 1;  /*do this because not allowed to move data within y */
   m = 1;
   rzpowr( n, &m );

   if( *n ¡= 0 )  /* trouble from rzpowr */
 return( -1 );

   if( *n ¿ nhold )-  /* arrays aren't big enough for the job */
 *n = -*n;
 return( -1);
   "
   if( *newpk ¿ 1)-  /* trans is loaded. Need to keep same size. */
 if (*n ¿ *newpk) -
*n = -*n;
return( -2);
 "
 *n = *newpk;
   "

/*  BEGIN code for cos-fill  PART 2/2
*   This is the code for a cos-fill.  If you use this fill method,
*   replace part 2 of the flip-fill code in this function
*   with the code shown here:
*/
/*
   arg = 3.14159/(*n-n2+n1-1);
   for (i = n2+1; i ¡ *n; i++)-
y[i] = y[n2] + (y[n1]-y[n2])*(1.0 - cos( arg*(i-n2) ))/2;
```

```
   "
   for (i = 0; i¡n1; i++)-
y[i] = y[n2] + (y[n1]-y[n2])*(1.0 - cos(arg*(i+*n-n2)))/2;
   "
   *frac = (double)(n2 - n1 + 1)/(double)*n;
*/
/*  END code for cos-fill PART 2/2
*/

/*  BEGIN code for spline-fill  PART 2/2
*  This is the code for a spline-fill. If you use this fill method,
*  replace part 2 of the cos-fill or flip-fill code in this function
*   with the code shown here:
*/
   *frac = (double)(n2-n1+1) / (double)*n;
   nwidth = MAX( (nfwhm/2), 3 );
   z[0] = n2;
   z[2] = n1 + *n;
   z[1] = (z[0] + z[2])/2.0F;
   slout = 0.0;
   slin = 0.0;
   a[0] = 0.0F;
   a[1] = 0.0F;
   for (i=0; i¡ nwidth; i++)-
a[0] += y[n2 -i];
slout += y[n2 - i - nwidth];
a[1] += y[n1 + i];
slin += y[n1 + i + nwidth];
"
   slout = a[0] - slout;
   slin = slin - a[1];
   slout /= (double)nwidth*nwidth;
   slin /= (double)nwidth*nwidth;
   a[0] /= (float)nwidth;
   a[2] = a[1]/(float)nwidth;
   a[1] = (a[0] + a[2])/2.0F;
   rzspbs( z,a, 3, slout, slin, b, u);
   for( i = n2 + 1 ; i ¡ *n; i++ )-
rzspbt( z, a, b, 3, (double)i, &arg );
y[i] = (float)arg;
   "
   for( i = *n ; i ¡= *n + n1 - 1; i++ )-
```

```
rzspbt( z, a, b, 3, (double)i, &arg );
y[i - *n] = (float)arg;
   "
/*  END code for spline-fill PART 2/2
*/

/*  BEGIN code for flip-fill  PART 2/2
*   This is the code for a flip-fill. If you use this fill method,
*   replace part 2 of the cos-fill code in this function
*   with the code shown here:
*/
/*
  *frac = (double)(n2-n1+1)*2/(double)*n;
  ncent = (n2 + n1)/2;
  for( i = 0; i ¡ n1; i++ )-
y[i] = y[ncent];
  "
  for( i = n2 + 1 ; i ¡ *n; i++ )-
y[i] = y[ncent];
  "
  for( i = 1; i ¡= ((n2 - n1 + 1)/2); i++ )-
if( (n1 - i) ¿= 0 )-
   y[n1 - i] = y[n1 + i];
"
else-
   y[*n  + n1 - i] = y[n1 + i];
"
if( (n2 + i) ¡= (*n - 1) )-
   y[n2 + i] = y[n2 - i];
"
else-
   y[i - *n + n2 ] = y[n2 - i];
"
  "
*/
/*  END code for flip-fill PART 2/2
*/
  return(0);
"
/* ************************************************************************/
/* ************************************************************************/
```

11.3 rzparm

If your peakshapes are symmetric, you should use this function as given in **rzrserve.c**, without further ado. However, when one has asymmetric peakshapes, the question of how to identify the center becomes important. Is the center at the highest point, or at the center of mass? **rzparm** in **rzrserve.c** uses a center-of- mass criterium for setting the center (fiducial point) of the peakshapes.

An alternate way to calculate the fiducial point is shown in the form of **rzparm** given below. If you plan to use this alternate form, be sure that your peakshapes are smooth. A highest-point criterium is easily undermined by noise.

```
/* ********************************************************************

void rzparm(float shape[],long *nc,long *nh,double *pkhite,long *nfwhm,
long *minwid, long nl1, long nl2, long newpk)

* rzparm(shape,nc,nh,height,nfwhm,minwid,nl1,nl2,newpk)      *
*     *
* when newpk ¿ 1,no action by rzparm,      *
* assume parms have already been calculated      *
*     *
* when newpk = 1:      *
* a NEW peakshape is presented in array shape,   *
* shape is filled between data points nl1 and nl2      *
*     *
* rzparm calculates center-of-mass of peak = nc      *
*   highest point in peak = nh      *
*     full-width-at-half-max = nfwhm      *
* 2*(minimum half-width) = minwid      *
* (minwid = nfwhm for symmetric shapes)      *
* height of peak = pkhite      *
*     *
* fiducial point nc = center of mass of peak      *
* user may wish to calculate nc in alternate fashion      *
********************************************************************
/* rzparm uses Razor Library functions
rzamin
rzacon
rzanor
*/
#pragma comment(exestr," (c) Copyright 1991-92 Spectrum Square Associates, Ir
void rzparm(float shape[],long *nc,long *nh,double *pkhite,long *nfwhm,
```

```
long *minwid, long nl1, long nl2, long newpk)
-
   long i, l1, l2, negpos;
   float half, halfarea, height, ebase, adjust, yvalue;
   double shmin, shmax, shsum;
 double done;

 if( newpk ¿1 ) return;

   done = 1.0;
   rzamin( shape, &shmin, nl1,nl2 );
   rzamax( shape, &shmax, nl1,nl2 );
   negpos = 1;
   half = shape[nl1] + shape[nl2] - (float)(shmin*2);
 half = (float)fabs( (double)half );
   halfarea = (float)(shmax*2) - shape[nl1] - shape[nl2];
   halfarea = (float)fabs( (double)halfarea );
   if( half ¿ halfarea )-
negpos = -1;
rzmcon( shape, -done, nl1, nl2 );
shmin = - shmax;
"
   rzasum( shape, &shsum, nl1,nl2 );

 ebase = shape[nl1] + shape[nl1 + 1] + shape[nl1 + 2];
 ebase += shape[nl2 - 2] + shape[nl2 - 1] + shape[nl2];
 ebase /= 6.0F;
 half = ebase*(nl2 - nl1 + 1);
 if( shsum ¿ half )-
adjust = 1.0F;
"
 else-
adjust = 0.0F;
"
 shsum = shsum - half*adjust;

 halfarea = (float)(shsum/2);
   half = 0.0F;
   height = 0.0F;

   for( i = nl1; i ¡= nl2; i++ )-
yvalue = shape[i] - ebase*adjust;
```

```
if( yvalue ¿= height )-
height = yvalue;
*nh = i;
"
  half += yvalue;
  if( half ¡ halfarea )
    *nc = i + 1;
    "

 /* SOMETIMES NC IS TOO BIG - WHEN THERE IS LOTS OF NOISE, AND
  * WHEN THE BASELINE IS CURVED */
 if( *nc == nl2 + 1 )
*nc = *nh;
   i = nl1;
   l1 = i;
 yvalue = height/2.0F + ebase*adjust;
 while( shape[i] ¡= yvalue && i ¡ nl2) -
l1 = i;
i += 1;
"

   i = *nh + 1;
   l2 = i;
   while( shape[i] ¿ yvalue ) -
l2 = i;
i += 1;
"

   *minwid = MIN( (*nc - l1), (l2 - *nc) )*2;
   *nfwhm = MAX( (l2 -l1), 1 );
   *pkhite = height * negpos;
   if( negpos == -1 )
rzmcon(shape, -done, nl1, nl2);
   return;
"
/* ************************************************************************
```

11.4 rzsizn tells array sizes.

```
/* ************************************************************************
/*
```

```
long rzsizn(long n2,float shape[],long nl2);
long rzsizn(long,float[],long);
* The function rzsizn tells size needed for arrays.
* (input) n2 = last index of data, (i.e., ydata has size n2+1)
* (input) shape = shape array containing peakshape
*  which is needed to calculate nfwhm
* (input) nl2 = index of last data point in shape
* (output)  rzsizn  = minimum size needed for arrays
*/
/* ************************************************************************
#pragma comment(exestr," (c) Copyright 1991-92 Spectrum Square Associates, I
long rzsizn(n2, shape, nl2)
long n2;
float shape[];
long nl2;
-
double height;
long n, nh, nfwhm, minwid, newpk;
newpk = 1;
rzparm(shape,&n,&nh,&height,&nfwhm,&minwid,0,nl2,newpk);

/* BEGIN code for cos-fill and for spline-fill
* This is the code for a cos-fill. If you use this fill method, replace
* the flip-fill code in this function with the code shown here:
*/
n = n2 +1 +nfwhm*3;
/* BEGIN code for flip-fill
* This is the code for a flip-fill. If you use this fill method, replace
* the cos-fill code in this function with the code shown here:
*/
/*
 n = (n2 + 1)*2;
*/
/* END code for flip-fill
*/

/* Addition 1/21/96 - guard against cases where nl2 ¿¿ n2! */
if( (nl2 + 1) ¿ n )
     n = nl2 + 1;
nh = 1;
rzpowr(&n,&nh);
return(n);
```

```
" /* end of function */
/*************************************************************************
```

11.5 Error messages from rzrerr.

Use the function **rzrerr** to print out the error text for any function which returns a value < 0. Translate the error text into another language if needed.

```
/* ***********************************************************************
*   rzrser23.c
* ***********************************************************************
/* ***********************************************************************

#include ¡string.h¿

#include "razor.h"
/* ***********************************************************************
#pragma comment(exestr," (c) Copyright 1991-96 Spectrum Square Associates, Inc.'
void rzrerr( ierror, errtxt )
long ierror;
char *errtxt;
-
char *errlist[20] = -
"RAZOR LIBRARY 3.0 (C)1991-96 Spectrum Square Associates Inc.",
"Array sizes are not big enough for Fourier transforms.",
"Requested transform size ¿ previous size. Set newpk=1.",
"Newpk must contain size of trans array.",
"Baseline function error. (kmax¡1)",
"Peak shape too narrow. Use finer data/peakshape sampling.",
"# variables exceeds # datapolongs.",
"# variables exceeds matrix dimensions.",
"Error in peak type input.",
"Peak type not yet implemented.",
"Unable to achieve better chisq.",
"Unstable. Try different shapes. Did you miss a peak?",
"Degenerate. Did you place two peaks too close together?",
"Covariance matrix has negative diagonals. Suggest more iters.",
"Peak shape too wide for this function.",
"Too many negative data polongs. Need positive data.",
"Unable to establish RMS noise value. Peakshape too narrow?",
"Solution did not converge.",
```

```
"Only File-based baseline (background) permitted in bunch mode.",
"Function error. Contact Spectrum Square Associates, Ithaca NY, USA" ";

if( ierror != 0 )
ierror = - ierror;
if( ierror > 19 )
ierror = 19;
strcpy( errtxt, errlist[ierror] );
return;
"
/* ****************************************************************** */
/* ****************************************************************** */
*   END module rzrser23     *
** ***************************************************************** */
/* ****************************************************************** */
/* end modules     */
/* ****************************************************************** */
/* ****************************************************************** */
*   end of rzrserve.c
* ***************************************************************** */
```

11.6 rzpkst - Sorts peaks from rzrpic/rzrbas

```
/*****************************************************************
/*              BEGIN module rzrser25.c
 ***************************************************************** */
/* ****************************************************************** */
/* ****************************************************************** */
/*
void rzpkst(long n, long locpks[], long nloc, float sigpks[], long nsig,
long iway, long itest)

* sorts arrays locpks[0,...n-1], sigpks[0,..n-1], sigpks[nloc,..nloc+n-1]
* IF IWAY=1, SORTS UP
* IF IWAY=-1, SORTS DOWN
* IF ITEST=1, SORTS ON LOCPKS
* IF ITEST=2, SORTS ON SIGPKS
* IF ITEST=3, SORTS ON SIGPKS(NLOC,...)
* IF ITEST=4, SORTS ON SIGPKS(NLOC*2,...)
*/
/* ****************************************************************** */
```

```
void rzpkst(long n,long locpks[],long nloc,float sigpks[],long nsig,
long iway,long itest)
-
long i, j, nsort;
float a, atest, b, c, d, octest;

nsort = n;
if( nsort ¿ nsig )
nsort = nsig;
if( nsort ¿ nloc )
nsort = nloc;
for( j = 1; j ¡ nsort; j++ )-
a = locpks[j];
b = sigpks[j];
if( nloc + j ¡ nsig )
c = sigpks[nloc + j];
if( nloc*2 + j ¡ nsig )
d = sigpks[nloc*2 + j];
for( i = j - 1; i ¿= 0; i-- )-
atest = a;
octest = locpks[i];
if( itest == 2 )-
atest = b;
octest = sigpks[i];
"
if( itest == 3 && nsig ¿= nloc*2 )-
atest = c;
octest = sigpks[nloc + i];
"
if( itest == 4 && nsig ¿= nloc*3 )-
atest = d;
octest = sigpks[nloc*2 + i];
"
if( iway ¿ 0 && octest ¡= atest )
goto L_10;
if( iway ¡ 0 && octest ¿= atest )
goto L_10;
locpks[i + 1] = locpks[i];
sigpks[i + 1] = sigpks[i];
if( nloc + i + 1 ¿= nsig )
goto L_11;
sigpks[nloc + i + 1] = sigpks[nloc + i];
```

```
if( nloc*2 + i + 1 ¿= nsig )
goto L_11;
sigpks[nloc*2 + i + 1] = sigpks[nloc*2 + i];
L_11:
;
"
i = -1;
L_10:
locpks[i + 1] = (long)(a);
sigpks[i + 1] = b;
if( nloc + i + 1 ¿= nsig )
goto L_12;
sigpks[nloc + i + 1] = c;
if( nloc*2 + i + 1 ¿= nsig )
goto L_12;
sigpks[nloc*2 + i + 1] = d;
L_12:
;
"
return;
" /* end of function */
/*****************************************************************
```

11.7 rzdfil - Loads peaks from rzrpic/rzrbas into datmat

```
/*****************************************************************
/*              BEGIN module rzrser25.c
*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*
void rzpkst(long n,long locpks[],long nloc,float sigpks[],long nsig,
long iway,long itest)

* sorts arrays locpks[0,...n-1], sigpks[0,..n-1], sigpks[nloc,..nloc+n-1]
* IF IWAY=1, SORTS UP
* IF IWAY=-1, SORTS DOWN
* IF ITEST=1, SORTS ON LOCPKS
* IF ITEST=2, SORTS ON SIGPKS
* IF ITEST=3, SORTS ON SIGPKS(NLOC,...)
* IF ITEST=4, SORTS ON SIGPKS(NLOC*2,...)
```

```
*/
/* ****************************************************************************
void kzpkst(long n,long locpks[],long nloc,float sigpks[],long nsig,
long iway,long itest)
-
long i, j, nsort;
float a, atest, b, c, d, octest;

nsort = n;
if( nsort ¿ nsig )
nsort = nsig;
if( nsort ¿ nloc )
nsort = nloc;
for( j = 1; j ¡ nsort; j++ )-
a = locpks[j];
b = sigpks[j];
if( nloc + j ¡ nsig )
c = sigpks[nloc + j];
if( nloc*2 + j ¡ nsig )
d = sigpks[nloc*2 + j];
for( i = j - 1; i ¿= 0; i-- )-
atest = a;
octest = locpks[i];
if( itest == 2 )-
atest = b;
octest = sigpks[i];
"
if( itest == 3 && nsig ¿= nloc*2 )-
atest = c;
octest = sigpks[nloc + i];
"
if( itest == 4 && nsig ¿= nloc*3 )-
atest = d;
octest = sigpks[nloc*2 + i];
"
if( iway ¿ 0 && octest ¡= atest )
goto L_10;
if( iway ¡ 0 && octest ¿= atest )
goto L_10;
locpks[i + 1] = locpks[i];
sigpks[i + 1] = sigpks[i];
if( nloc + i + 1 ¿= nsig )
```

```
/*
*************************************************************************
*           rzrser24.c
* ************************************************************************
*/
/* Copyright (c) 1991-2002 Spectrum Square Associates, Inc., Ithaca NY  14850
    All rights reserved.
*/
/* ************************************************************************
*/
#include <math.h>

#include "razor.h"
/* ************************************************************************
*/
/* ************************************************************************  *
*        Rzdfil fills the data matrix datmat[] needed for RazorFit.         *
*           using the output of the peak-picker, plus info about
*           the desired peak type, and desired baseline type.
*           NOTE: rzdfil always uses the first naccept peaks in locpks,
*               and sets all peaks to same type.
*
*   Input: datmat[][40], dimensioned to datmat[naccept+2][40] if basetype=0,
*                        else dimensioned to datmat[naccept+3][40]
*           locpks from rzrpic or rzrbas
*           npick = number of peaks found by rzrpic or rzrbas
*           sigpks from rzrpic or rzrbas
*           nsig = size of sigpks, as per rzrpic, rzrbas
*           naccept = number of peaks accepted(first naccept peaks of
locpks..)
*           nbunch = max number of peaks to be processed in each bunch.
*               if nbunck = 0, ALL peaks will be processed simultaneously
*           peaktype = type from peak catalog
*           ffixc, ffixh, ffixw, ffixa = fix\vary flags for center, height,
*                                                 width, 4th (asymmetry)
parm
*                                       = 0 for fixed, 1 for variable
*           basetype = baseline type ( 0 = no baseline )
*                       = chosen from baseline catalog.
*           ffixb0, ffixb1, ffixb2, = fix/vary for baseline parms
*                                                  = 0 for fixed, 1 for
variable.
*           ymin = minimum value in data (used to set up baseline parms)
*           xstart, xstep = starting x-value, x-longerval, (user-coordinates)

*               of first data point in data which will be processed
*
* ************************************************************************
*/
/* void rzdfil(float datmat[][40],long locpks[],long npick,float
sigpks[],long nsig,
long naccept,long nbunch,long peaktype,
long ffixc,long ffixh,long ffixw,long ffixa,
long basetype,long ffixb0,long ffixb1,long ffixb2,double ymin,
double xstart,double xstep) */
EXPORT32 void FAR EXPORT rzdfil(float  datmat[][40],long FAR *locpks,
long npick,float FAR *sigpks,long nsig,
long naccept,long nbunch,long peaktype,
long ffixc,long ffixh,long ffixw,long ffixa,
long basetype,long ffixb0,long ffixb1,long ffixb2,double ymin,
double xstart,double xstep)
{
    long npks, i;
/*
*                       about filling datmat
*
* *  DATMAT will have naccept+2 rows.  Each row of DATMAT has 40 positions.
*    In DATMAT, naccept is the number of peaks, including the baseline, if
```

```
any.

* * The FIRST ROW of DATMAT looks like this:
*       datmat[0][0]=npks
*       datmat[0][1]=nbunch
*       datmat[0][2]=bunch flag
*       datmat[0][3]=iter
*       datmat[0][4]=reduced chisq
*       datmat[0][5]=chitest
*       datmat[0][6]=cnvg
*       datmat[0][7]=cnvgtest
*       datmat[0][8]=USED TO FORCE A USER-DEFINED SCALING, IF NOT ZERO
*       datmat[0][9]=J1, first peak of current bunch
*       datmat[0][10]=J2, last peak of current bunch
*       datmat[0][11]=reserved for rzupdt (xstart)
*       datmat[0][12]=reserved for rzupdt (xstep)
*
*
*      The first row of DATMAT transmits and receives program control data.
*      If you wish to process the peaks all-at-once,
*       only the first position datmat[0][0] must be filled on input.
*      If you wish to process the peaks in bunches,
*       fill datmat[0][0], datmat[0][1], datmat[0][2].
*        Set everything else to 0.0!
*           except that you are allowed to usurp datmat[0][11] and
datmat[0][12]
*           for your own use.  One possible use is in rzupdt.  Rzupdt
*           set limits in data-point coordinates.  But you can use
*           xstart and xstep to set limits in user-coordinate.
*/
     /* Filling the DATMAT control vector */
     npks = naccept;
     if( basetype > 0)
        npks += 1;
    datmat[0][0] = (float)npks;      /* Number 'peaks', including baseline */
    datmat[0][1] = (float)nbunch;
     if( nbunch > 0 ){
        datmat[0][2] = 1.0F;
        }
     else{
        datmat[0][2] = 0.0F;
     }
     datmat[0][3] = 0.0F;   /* Iter: You MUST set iter=0 for start!!!! */
    datmat[0][4] = 0.0F;   /* And set everything else to zero too. */
    datmat[0][5] = 0.0F;
    datmat[0][6] = 0.0F;
    datmat[0][7] = 0.0F;
    datmat[0][8] = 0.0F;
    datmat[0][9] = 0.0F;
    datmat[0][10] = 0.0F;
    datmat[0][11] = (float)xstart;  /* Fill and use as desired. */
    datmat[0][12] = (float)xstep;   /* Fill and use as desired. */
    datmat[0][13] = 0.0F;
    datmat[0][14] = 0.0F;
    datmat[0][15] = 0.0F;
    datmat[0][16] = 0.0F;
    datmat[0][17] = 0.0F;
    datmat[0][18] = 0.0F;
    datmat[0][19] = 0.0F;


/*
*      OUTPUT: DATMAT uses the first row to keep track of what it is doing
*              from one iteration to the next.  In this way, we can return
*              control to you so that you may display output each
*              iteration if you wish.  Most likely, you will only be
*              longerested in the values of ITER and CHISQ.
*
```

```
* * SUBSEQUENT ROWS of DATMAT:
*    Each subsequent row of DATMAT delivers and returns data about ONE peak.
*    In this example program, for simplicity we assume all peaks will be of
*    the same type.  This is certainly not necessary or even usually
desirable.
*
*    On input, you will fill 11 of the 40 positions:
*      type, c, fixc, h, fixh, w, fixw, a, fixa, p, fixp, [  ], [   ]
*
*    On output, you will be longerested in all 40 entries:
*      type, c, errc, h, errh, w, errw, a, erra, p, errp, area, errarea
*
* *   Modify the loading procedure to suit your needs.
*
*************************************************************************
*
*                  --   HERE IS HOW INPUT IS CARRIED OUT  --
*
*  *   TYPE   The user selects the type of each peak from the list below.
*  *   C      C = estimated center position.
*             In this example, the center comes from RZRPIC (in LOCPKS).
*  *   H      H = estimated height.
*  *   W      W = estimated width.
*             The initial height and width of each peak is set
*             using information from RZRPIC, stored in SIGPKS.
*  *   A      A = fourth parameter (often an asymmetry).
*             Fourth parameters are initialized after the TYPES are known.
*  *   P      P = fifth parameter.
*             If you have written a peak function which has a fifth
parameter,
*             then you must set initial values for P.
*
*  *   fixc, fixh, fixw, fixa, fixp
*             fix = 0 for parameters which are fixed (not variable).
*             Often, you will want to constrain certain peakshape parameters
*             to be positive.  In this example, we will constrain all
*             shape parameters to be positive.
*
*      The rules are as follows:    (See function rzupdt in rzserve.c for the
*                                        code that enforces them).
*
*          fix = 0  =>  Initial value of parameter never changes
*          fix > 0  =>  Parameter > 0.
*          fix < 0  =>  Parameter may have either sign
*          fix = +1 =>  0.0 <= parameter <= infinity
*          fix = -1 =>  -inf <= parameter <= inf
*          abs(fix)=2  Asymmetry parameter DATMAT(8,I) is bounded in rzupdt
*                      by fixed limits (0,1) or (-1,+1)
*          abs(fix)=3  Width or position parameter (DATMAT(6,I) or
DATMAT(2,I)
*                      is bounded in rzupdt
*
*  *   Of course, your spectrum may not be the sum of only positive definite
*  *   shapes; if not, then be sure to set FIX=<0 for those heights which
*  *   may be negative (an example is a derivative spectrum). Widths (and
*  *   mixing coefficients for, e.g., Gauss-Lorentz sum peaks), must be
*  *   positive; for these, set FIX=>0.
*  *   If a shape parameter should NOT be altered by the program, set FIX=0.
*          Of course, you may set up your own rules by modifying rzrupdt.
*/

/*
*       LOADING DATMAT(*,2...NACCEPT)
*       Now load in the parameters from RZRPIC to furnish
*       default estimates of position, width, mixture fraction of peak:
*/
```

```
     /* must sort locpks and sigpks to load peaks in order of increasing
        position (in locpks), if wish to process in bunch-mode
     */
     if( nbunch > 0 )
        rzpkst( naccept, locpks, npick, sigpks, nsig, 1L, 1L );

    for( i = 1; i <= naccept; i++ ){
         datmat[i][0] = (float)peaktype;                /* peak types */
         datmat[i][1] = (float)locpks[i - 1];           /* positions, from
         RZRPIC */
         datmat[i][2] = (float)(-3.0*ffixc);       /* for bounded positions
         */
           if( datmat[0][2] > 0. )
                datmat[i][2] = (float)(-4.0*ffixc);
         datmat[i][3] = sigpks[npick + i - 1];   /* heights */
         datmat[i][4] = (float)(1.0*ffixh);        /* for positive heights */
// testing
//datmat[i][4] = (float)(9.0*ffixh);        /* for positive heights */
//datmat[i][23] = datmat[i][3]/2.0;
//datmat[i][24] = datmat[i][3]*2.0;
// end testing
         if( datmat[i][3] < 0.0F )                    /* for negative heights
        */
                datmat[i][4] = (float)(-1.0*ffixh);

            /* TESTING - FOR 2D CODE */
            if( locpks[i-1] < 0.0F ){
                datmat[i][19] = -1.0F;
                datmat[i][1] = -(float)locpks[i-1];
                datmat[i][4] = 0.0F;
                }
            else{
                datmat[i][19] = 1.0F;
                }
            /* END TESTING - FOR 2D CODE */

            datmat[i][5] = sigpks[npick*2 + i - 1];  /* widths */
            datmat[i][6] = (float)(3.0*ffixw);         /* for bounded widths
            > 0  */
            datmat[i][7] = 0.0F;
            datmat[i][8] = 0.0F;
            datmat[i][9] = 0.0F;
            datmat[i][10] = 0.0F;
         datmat[i][11] = 0.0F;
         datmat[i][12] = 0.0F;
         datmat[i][13] = 0.0F;
         datmat[i][14] = 0.0F;
         datmat[i][15] = 0.0F;
         datmat[i][16] = 0.0F;
         datmat[i][17] = 0.0F;
         datmat[i][18] = 0.0F;
         datmat[i][19] = 0.0F;
         if( peaktype == 0 )
                    datmat[i][8] = (float)(-2.0*ffixa);
            if( peaktype == 3 ){
            datmat[i][7] = .5F;                  /* mixing parm for Gauss/Lrnz
            */
              datmat[i][8] = (float)(2.0*ffixa);
          }
            if( peaktype == 4 ){
            datmat[i][5] *= 2.0F;
              datmat[i][7] = datmat[i][5];
              datmat[i][8] = (float)(3.0*ffixa);
          }
            if( peaktype == 5 || peaktype == 6 ){
                datmat[i][7] = 0.0F;
              datmat[i][8] = (float)(-2.0*ffixa);
          }
```

```
            if( peaktype == 7 ){
              datmat[i][7] = 1.0F;
              datmat[i][8] = (float)(1.0*ffixa);
              datmat[i][9] = 0.0F;
              datmat[i][10] = (float)(-1.0*ffixa);
          }
            if( peaktype == 8 ){
              datmat[i][7] = 2.0F;
              datmat[i][8] = (float)(1.0*ffixa);
          }
            if( peaktype == 9 ){
              datmat[i][3] = 0.1F;
              datmat[i][4] = 1.0F;
              datmat[i][5] = sigpks[npick*2+i-1];
              datmat[i][6] = (float)(3.0*ffixw);
              datmat[i][7] = sigpks[npick*2+i-1];
              datmat[i][8] = (float)(3.0*ffixw);
              datmat[i][9] = (float)sqrt( (double)(sigpks[npick*2+i-1]/(16.0*
              3.1416*locpks[i-1])) );
              datmat[i][10] = 1.0F;
              datmat[i][11] = 1.0F;
              datmat[i][12] = 1.0F;
            }
            if( peaktype == 10 ){
              datmat[i][7] = sigpks[npick*2+i-1];
              datmat[i][8] = (float)(3.0*ffixw);
          }
         if( peaktype >= 100 && peaktype < 200)
                    datmat[i][8] = (float)(-2.0*ffixa);
    }
    if( basetype == 200 ){
        datmat[npks][0] = (float)basetype;
        for( i = 1; i <= 19; i++ ){
        datmat[npks][i] = 0.0F;
        }
    }
   else if( basetype >= 201 && basetype <= 204 ){
        datmat[npks][0] = (float)basetype;
     datmat[npks][1] = (float)ymin;
       datmat[npks][2] = (float)(-1.0*ffixb0);
       for( i = 3; i <= 19; i++ ){
         datmat[npks][i] = 0.0F;
         }
       if( basetype > 201 )
            datmat[npks][4] = (float)(-1.0*ffixb1);
       if( basetype > 202 )
         datmat[npks][6] = (float)(-1.0*ffixb2);
    }
     else{
        ;
     }
    return;
}
/* ***************************************************************************
*     END module rzrser24                                                    *
** ***************************************************************************
*/
```

```
goto L_11;
sigpks[nloc + i + 1] = sigpks[nloc + i];
if( nloc*2 + i + 1 ¿= nsig )
goto L_11;
sigpks[nloc*2 + i + 1] = sigpks[nloc*2 + i];
L_11:
;
"
i = -1;
L_10:
locpks[i + 1] = (long)(a);
sigpks[i + 1] = b;
if( nloc + i + 1 ¿= nsig )
goto L_12;
sigpks[nloc + i + 1] = c;
if( nloc*2 + i + 1 ¿= nsig )
goto L_12;
sigpks[nloc*2 + i + 1] = d;
L_12:
;
"
return;
" /* end of function */
/*****************************************************************
```

11.8 rzrxpk - Removes baseline, smooths peakshape

rzrxpk helps in extracting a peakshape out of a data file. The baseline is automatically removed, and both a baseline-corrected peakshape and a smoothed, baseline-corrected peakshape are made available.

```
/* ****************************************************************
/*     Razor X Peak
long rzrxpk(float shape[],long nl2,float y[],float w[],float x[],long n,
double *bsens,long *nfwhm,double *sigma);
*
*   EXTRACTS A PEAKSHAPE BY REMOVING BASELINE AND (OPTIONALLY) SMOOTHING
*
*     INPUT:
*          SHAPE IS THE INPUT DATA, CONTAINING PEAKSHAPE
*          NL2 is the index of the final point in shape(0,nl2)
*          X,W,Y = WORK ARRAYS
```

```
*          N = is the size of the work arrays. Require N ¿= NL2+1
*          BSENS = baseline sensitivity (Try *bsens = 1.0)
*
*     OUTPUT:
*          Y = XTRACTED PEAK
*          W = SMOOTHED-EXTRACTED PEAK
*          X = BASELINE
*          NFWHM = WIDTH OF PEAK
*          SIGMA = RMS NOISE IN DATA
*
*/
/* ************************************************************************
```

11.9 New peakshapes

RazorFit gives you a fairly good selection of peakshapes, but we recognize that there are many others we have left out. If you need something else, follow the templates, and fill up the empty peakshape functions of **rzrserve.c**. RazorFit is set up to call them.

The peakshape functions are not particularly fast. They generate the peak models by making a separate call for each data point. This allows RazorFit to do its calculations without allocating many more arrays. It was a matter of trading time for space. The faster method requires *many* work arrays, each the same size as the original data set. The number of ADDITIONAL work arrays needed is equal to the number of parameters in the model!

If your data sets are small, or if you have lots of space, we encourage you to call us. There is always a Next Time....

13 Nov. 2003 **QPeaks Documentation**

QPEAKS

QPeaks was developed for finding peaks in mass spectrometry, but it can be used for any files containing peaks. QPeaks uses Razor Library's peak-picking function *rzrpic* to find peaks in a spectrum. It also calls upon Razor's peak-fitting function *rzrfit* to find the best-fitting parameters (position, height, width) of the identified peaks.

QPeaks is faster and more flexible to use than standard Razor Library functions. It is faster because it processes a spectrum by sections, rather than all-at-once. It is more flexible because it handles data from (a) spectrometers that operate at constant peak width (Δm) and (b) spectrometers that operate at constant resolution (m/Δm). (Razor Library peak-finding functions all operate in constant width mode).

QPeaks is easier to use than the Razor Library because all the programming details are automatically handled behind-the-scenes within the DLL. Qpeaks programming calls are much easier to implement than the programming calls for Razor Library functions.

QPeaks is easily tailored for a particular spectrometer by specifying the following:

1. Peak Width or Resolution. QPeaks requires that the user identify an approximate width for the peaks in the data. It performs in two modes: (a) constant peak width, and (b) constant resolution or resolving power (i.e. m/Δm for mass spectrometry).
2. Signal/Noise cutoff for peak-finding.
3. Number of the *rzrpic* peak-picker to use. (In practice, we have found that picker #4 seems to be the best performer for the mass spectrometry files we have dealt with so far, and so it is usually selected at the start.).
4. Noise Statistics (Poisson or Normal/Gaussian).
5. For Normal noise, RMS Noise if known (else input zero).
6. For Poisson noise, number of scans that were averaged, if known (else input zero).
7. Baseline parameter, if baseline removal is desired.

Thus one can select peaks by signal/noise ratios in either gaussian or poisson noise environments, and tune QPeaks performance by choosing different peak pickers.

Overview: RazorQPK and RazorDQPK DLLs

RazorQPK contains a single-precision implementation of Qpeaks; RazorDQPK contains a double-precision implementation.

The purpose of the QPeak (QPK) algorithm of the RazorQPK DLL is to find peak positions, and to perform a Levenberg-Marquardt fit of those peaks to find the most probable (maximum likelihood) peak positions, heights, widths, and areas. QPK also has an option to perform automated baseline finding in conjunction with the peak finding.

QPK output includes (1) a table listing the found peaks and their parameters, and (2) an array showing the positions and amplitudes of the found peaks, as well as the found baseline (if requested). Other arrays are also available; see the Qpkqpk.cpp source file.

Detail: Input

The RazorQPK DLL requires the following information for QPK processing:

- **xdata**, **ydata** arrays of length **numdata**, containing mass spectrometer m/z data. RazorQPK assumes that the spectrometer data is complete (i.e. no missing data samples, profile data).

- **spectrometer (singlet) width**. The spectrometer smearing width is the full-width at the half-power point (FWHM) of typical isolated peaks in the spectrometer data. This width may be the same as the instrumental resolution; it may be the natural width of peaks. Choose whichever width *matches* the widths of peaks as seen in the data. Units are the same as the units of the x_axis of the data (usually m/z).

- **peak_signal2noise**. Cutoff for the peak finder. (Same as psens for *rzrpic*. See the Razor Library manual.)

  The peak finder operates on a second derivative of the data. The finder estimates the height and width and area of a peak from the shape of the second derivative, using the input peak shape to assist in this task. It then accepts or rejects the candidate peak according to the peak_signal2noise criterion provided by the user.

  When the peak_signal2noise is positive, the acceptance is based on amplitude_ of_peak/amplitude_of_noise >= peak_signal2noise. When the peak_signal2noise is area, the acceptance is based on

area_of_peak/noise_area >= abs(peak_signal2noise). The noise_area is calculated as rms_noise*sqrt(peak_width_in_datapoints).

- **picker**. Same as iperf for *rzrpic*. See the Razor Library manual. The RazorQPK DLL contains an additional peak picker; number (iperf=) 10.
  Available Pickers: -1 Quick_Pick (single pass)
  -3 Quick_Pick (2 passes, narrow+wide)
  1 High_Performance
  2 High_Resolution
  3 2nd_Order High Performance
  4 Quiet_pick
  5 Narrow_Wide
  10 Perfect_10
  11 Gentle smooth, then Perfect_10
  Most of these pickers are 'resolving' pickers. When they encounter a peak that is wider than the spectrometer (singlet) width, they will attempt to resolve the wide component into peaks of the given width. If this performance is undesirable, choose one of the non-resolving Quick Pick (-1 or -3), or the semi-resolving Quiet Pick (4) picker.
  Most of these pickers will perform quite well even if the input spectrometer width (singlet width) is too wide (up to a factor of 1.5x or even 2x). Most will give too many peaks when the input spectrometer width is too narrow. Only the Quick Pick picker will forgive you when you give it too small a value for spectrometer width.
  If your singlet width is larger than 0.5amu, none of the pickers will find multiply-charged peaks (spaced by 0.5 amu). If you want to find doubly-charged peaks in a spectrum that has wide (>0.5 amu) peaks, set the singlet width to 0.5 amu, and use picker 4 or 11.

- **Noise_statistics**. If the noise statistics are normal/gaussian, or if you do not know what the statistics are, set noise_statistics = 1. If the noise statistics are Poisson, set noise_statistics = 2. Same as istat for *rzrpic*. See Razor Library manual. Default noise_statistics = 1.

  Recommendation: Usually, the noise statistics will be neither normal nor Poisson, but something in between. Our recommendation for in-between cases is this: If the noise on your biggest peaks is approximately the same as the noise on your smallest peaks, choose Normal (**noise_statistics** = 1), statistics, and set **rms_noise** =1. Then use **peak_signal2noise** to select all peaks with amplitudes greater than a particular value, i.e. set **peak_signal2noise** = 10 to select all peaks with amplitudes greater than 10. If the noise on your biggest peaks is larger than the noise on your smallest peaks, then your noise statistics are closer to Poisson. In this case, choose Poisson (**noise_statistics** = 2), and initially set **Poisson_scans** = 1.

Readjust **Poisson_scans** to a more appropriate value if necessary, but do **not** set Poisson_scans = 0.

rms_noise. The rms noise in the data (if known). If unknown, set ***input*** rms_noise=0.0, which is a signal for the DLL to estimate the rms noise. This parameter is only used for normal/gaussian statistics. Same as sigma for *rzrpic*. See Razor Library manual.

How does Qpeaks calculate the rms_noise? Rms_noise is returned from *rzrpic*. It is calculated inside rzrpic by first smoothing the data (usually via *rzresm*), and then by calculating the mean square difference between the smoothed and raw data.

Note: If you set ***input*** rms_noise=0.0, and your peaks are not well sampled (<3 points between half-power points), Razor may not be able to use rzresm to smooth the data. It will have to 'punt' (i.e. use a box-car smoothing algorithm). The smoothing will be too heavy-handed, the smoothed data will lose resolution, and Qpeaks will have trouble calculating an accurate value for rms_noise. QPeaks usually errs by returning too large a value for rms_noise in these cases.

- **Poisson_scans**. The number of scans that were averaged to obtain the current scale (in counts). If unknown, set ***input*** poisson_scans=0.0, which is a signal for the DLL to estimate this parameter. This parameter is only used for poisson statistics.

- **Baseline_width_multiplier**. The RazorQPK peak picker will automatically define a baseline (rzredg) under the data, before picking peaks. This usually helps in finding peaks in mass spectrometry data. Start with a value of the baseline_width_multiplier = 3. Increase this parameter if the baseline cuts too much energy out of the peaks. Set this parameter = 0 to turn off the automatic baseline.

  What does this baseline_width_multiplier actually do? If you look at the documentation for any of the baseline algorithms in the Razor Library manual, you will see that the algorithm want you to tell it the width (FWHM) of the *widest* peak in the data. It will then attempt to preserve any features that are equal to, or narrower than, the widest peaks. When Qpeaks calls upon one of the Razor Library algorithms for a baseline, it takes the singlet_width that you gave it, multiplies that width by the baseline_width_multiplier, and sends the resulting width into the baseline routine.

Readjust **Poisson_scans** to a more appropriate value if necessary, but do **not** set Poisson_scans = 0.

rms_noise. The rms noise in the data (if known). If unknown, set ***input*** rms_noise=0.0, which is a signal for the DLL to estimate the rms noise. This parameter is only used for normal/gaussian statistics. Same as sigma for *rzrpic*. See Razor Library manual.

How does Qpeaks calculate the rms_noise? Rms_noise is returned from *rzrpic*. It is calculated inside rzrpic by first smoothing the data (usually via *rzresm*), and then by calculating the mean square difference between the smoothed and raw data.

Note: If you set ***input*** rms_noise=0.0, and your peaks are not well sampled (<3 points between half-power points), Razor may not be able to use rzresm to smooth the data. It will have to 'punt' (i.e. use a box-car smoothing algorithm). The smoothing will be too heavy-handed, the smoothed data will lose resolution, and Qpeaks will have trouble calculating an accurate value for rms_noise. QPeaks usually errs by returning too large a value for rms_noise in these cases.

- **Poisson_scans**. The number of scans that were averaged to obtain the current scale (in counts). If unknown, set ***input*** poisson_scans=0.0, which is a signal for the DLL to estimate this parameter. This parameter is only used for poisson statistics.

- **Baseline_width_multiplier**. The RazorQPK peak picker will automatically define a baseline (rzredg) under the data, before picking peaks. This usually helps in finding peaks in mass spectrometry data. Start with a value of the baseline_width_multiplier = 3. Increase this parameter if the baseline cuts too much energy out of the peaks. Set this parameter = 0 to turn off the automatic baseline.

  What does this baseline_width_multiplier actually do? If you look at the documentation for any of the baseline algorithms in the Razor Library manual, you will see that the algorithm want you to tell it the width (FWHM) of the *widest* peak in the data. It will then attempt to preserve any features that are equal to, or narrower than, the widest peaks. When Qpeaks calls upon one of the Razor Library algorithms for a baseline, it takes the singlet_width that you gave it, multiplies that width by the baseline_width_multiplier, and sends the resulting width into the baseline routine.

area_of_peak/noise_area >= abs(peak_signal2noise). The noise_area is calculated as rms_noise*sqrt(peak_width_in_datapoints).

- **picker**. Same as iperf for *rzrpic*. See the Razor Library manual. The RazorQPK DLL contains an additional peak picker; number (iperf=) 10.
  Available Pickers: -1 Quick_Pick (single pass)
  -3 Quick_Pick (2 passes, narrow+wide)
  1 High_Performance
  2 High_Resolution
  3 2nd_Order High Performance
  4 Quiet_pick
  5 Narrow_Wide
  10 Perfect_10
  11 Gentle smooth, then Perfect_10
  Most of these pickers are 'resolving' pickers. When they encounter a peak that is wider than the spectrometer (singlet) width, they will attempt to resolve the wide component into peaks of the given width. If this performance is undesirable, choose one of the non-resolving Quick Pick (-1 or -3), or the semi-resolving Quiet Pick (4) picker.
  Most of these pickers will perform quite well even if the input spectrometer width (singlet width) is too wide (up to a factor of 1.5x or even 2x). Most will give too many peaks when the input spectrometer width is too narrow. Only the Quick Pick picker will forgive you when you give it too small a value for spectrometer width.
  If your singlet width is larger than 0.5amu, none of the pickers will find multiply-charged peaks (spaced by 0.5 amu). If you want to find doubly-charged peaks in a spectrum that has wide (>0.5 amu) peaks, set the singlet width to 0.5 amu, and use picker 4 or 11.

- **Noise_statistics**. If the noise statistics are normal/gaussian, or if you do not know what the statistics are, set noise_statistics = 1. If the noise statistics are Poisson, set noise_statistics = 2. Same as istat for *rzrpic*. See Razor Library manual. Default noise_statistics = 1.

  Recommendation: Usually, the noise statistics will be neither normal nor Poisson, but something in between. Our recommendation for in-between cases is this: If the noise on your biggest peaks is approximately the same as the noise on your smallest peaks, choose Normal (**noise_statistics** = 1), statistics, and set **rms_noise** =1. Then use **peak_signal2noise** to select all peaks with amplitudes greater than a particular value, i.e. set **peak_signal2noise** = 10 to select all peaks with amplitudes greater than 10. If the noise on your biggest peaks is larger than the noise on your smallest peaks, then your noise statistics are closer to Poisson. In this case, choose Poisson (**noise_statistics** = 2), and initially set **Poisson_scans** = 1.

Detail: Output

During setup for QPK processing, the programmer will have allocated two arrays, a workspace array, and an output array. (See QpkDirect.cpp, DqpkDirect.cpp).

The output array, which will be the same length as the input array, will contain spikes of appropriate height at the positions of the found peaks, superimposed on the found baseline (if requested).

A table of peak parameters is located within the workspace. This table is described in the next section. The peak table address is (float *)(Workspace + glthings[69]). The peak table address is (double *)(Workspace + glthings[69]) in RazorDQPK.

RazorQPK Peak Table

The RazorQPK DLL creates a table of peaks and peak parameters. This table, stored within the WorkSpace, is complete when the processing is complete (*percentDone = 100.0F). Instructions for accessing the table are given below. Example source code that accesses the table and prints it to a file is given in the files QpkDIrect.cpp and DqpkDirect.cpp.

The table has 1 row, 40 columns, for each peak. Columns are filled as follows:
1 Peak_ID Peak ID numbers are assigned sequentially.
2 Loc_Index The original index position of the peak, as chosen by *rzrpic*. This position is derived by looking at the second derivative of the data.
3 Total_Height Total height = height of peak + value in baseline array at position of peak center. RazorQPK processing automatically removes a baseline from the data, in order to obtain better parameters for the fitted peaks. The baseline is available in the workspace also, and may be displayed. The baseline address is (float *)(Workspace + glthings[62]).]).]) In RazorDQPK, the baseline address is (double *)(Workspace + glthings[62. The length of the baseline array is the same as the length of the raw data array.
4 RMS/Poiscns Value of RMS noise (for Normal statistics), or number of Poisson scans (for Poisson statistics), used in *rzrpic*.

[Columns 5-22 are results of peak fitting by rzrfit.]
5 m/z The center of mass of the peak, calculated by the Maximum Likelihood peak-fitting algorithm *rzrfit*. (Calculated from index_position in column 11).
6 +/-m/z Uncertainty in the peak center position. (Calculated from uncertainty in index_position, column 12).
7 peak_height The peak height returned from *rzrfit*, which performs a full Levenberg Marquardt fit.
8 +/-height Uncertainty in the peak height, calculated by *rzrfit*.
9 m/z_width Width of the peak. (Calculated from index_width in column 13).
10 +/-m/z_width Uncertainty in the peak width. (Calculated from uncertainty in index_position, column 14).

11 index_position Peak center, index location, as returned from *rzrfit*.
12 +/-index_posn Uncertainty in the peak index location, calculated by *rzrfit*.

The *rzrfit* peak-fitting procedure finds a best fit in the least-square sense for certain peak parameters (position, height, width, etc.) The +/- errors quoted for these parameters have the following meaning: If the measurement errors are independent, and are normally distributed, then the +/- errors given above are the 1-sigma width of a normal probability distribution for the corresponding parameter. (Columns 7-8,11-22 contain the output of *rzrfit*. Columns 5-6, 9-10 are derived from columns 11-14).

13 index_width Peak width, as returned from *rzrfit*.
14 +/-index_width Uncertainty in peak width, as returned from *rzrfit*.
15 rms_fit RMS difference between peak model and data in region of peak fit.
16 error_code Error code returned from *rzrfit*. Error code -10 indicates that the peak fit did not formally converge within the allowed 100 iterations; parameters must be taken with caution. Error code -13 means that the final matrix inversion for obtaining the rms errors of the parameters was ill-conditioned, and thus the errors are not to be trusted.
17 Area Peak area, (total counts), as returned from *rzrfit*. Note that even if the spectrum is a mass spectrum, with x_units Daltons, the area units reported in columns 17 and 18 will not be counts*Daltons. The area units for columns 17 and 18 are simply total counts.
18 +/- area Uncertainty in peak area (total counts), as returned from *rzrfit*.
19 Fourth parameter, as returned from *rzrfit*
20 +/- fourth parameter Uncertainty as returned from *rzrfit*.
21 Fifth parameter, as returned from *rzrfit*
22 +/- fifth parameter Uncertainty as returned from *rzrfit*.

23 First index of peak-picking segment
24 Last index of peak-picking segment.

[Columns 25-32 are outputs of the peak apex algorithm.]
25 Height of peak at apex position (derived from raw data).
26 Apex m/z position of peak (derived from raw data).
27 Apex m/z position of peak (derived from smoothed data).
28 Area (counts) of peak at apex position.
29 +/- Area (counts) of peak at apex position.
30 Start m/z for computing peak area given in column 28
31 End m/z for computing peak area given in column 28
32 RMS noise used for computing +/- area in column 29. This RMS value is obtained from the difference between the smoothed and raw data.
33-40 Reserved Columns 33-40 are currently used in testing and debugging.

Updates 20 Nov 2003:

1. Upgraded documentation in section 'Detail: Input' for parameters singlet_width, peak_signal2noise, baseline_width_mujltiplier, rms_noise.
2.

QPKDirect is furnished as
(a) **RazorQPK.DLL**, the DLL that performs peak processing of the data.
(b) **QPKDirect.exe,** a console program that drive the RazorQPK DLL.
(c) **QPKdirect.cpp**, source code for a demonstration console program that reads an XY data file, receives input from the user, processes the data, and writes output files. QPKDirect calls the RazorQPK DLL functions directly.
(d) **MOPread.cpp,** source code for functions to read/write ASCII XY data files.
(e) **Razor.h, MOPread.h, rzrqpk.h,** and **RazorQPK.lib.**

RazorQPK input requirements are:

- X and Y data arrays, containing the masses and counts from the spectrometer.
- Peak Width or Resolution. QPeaks requires that the user identify an approximate width for the peaks in the data. It performs in two modes: (a) constant peak width, and (b) constant resolution or resolving power (i.e. m/Δm for mass spectrometry).
- Signal/Noise cutoff for peak-finding.
- Number of the *rzrpic* peak-picker to use. In practice, we have found that picker #4, Quiet Pick, seems to be the best performer for the mass spectrometry files we have dealt with so far, and so it is usually selected at the start.
- Noise Statistics (Poisson or Normal/Gaussian).
- For Normal noise, RMS Noise if known (else input zero).
- For Poisson noise, number of scans that were averaged, if known (else input zero).
- Baseline parameter, if baseline removal is desired.

- DLL control parameter: length of time DLL is allowed to process before returning control to calling program (delayticks).

Using RazorQPK DLL - Method 2

Source code for the console program QPKDirect shows how to directly call functions within the RazorQPK DLL. The steps for directly calling the DLL are:

1. Obtain the required input parameters listed above.

2. Call CalcArraySizes to get the required size of the workspace array, and the required size of the output (result) array.

```
   EXPORT32 long FAR EXPORT CalcQpkArraySizes(
float *input_masses,float *input_intensities,long numdata,
float *input_spectrometer_shape_func,long numshape,
float singletwidth, float signal2noise, long iperf, long istat,
float sigma, float poiscns, float basewidthmult,
unsigned long *sizewkspace,long *numoutarray);
```

3. Allocate the workspace and the output array.

4. Call InitWorkSpaceQPK to initialize the workspace.

```
   EXPORT32 long FAR EXPORT InitWorkSpaceQpk(
float *input_masses,float *input_intensities,long numdata,
float *input_spectrometer_shape_func,long numshape,
float singletwidth, float signal2noise, long iperf, long istat,
float sigma, float poiscns, float basewidthmult,
char *WorkSpace,unsigned long sizeWorkSpace,
float *outputIntensities, long numOutputArray);
```

5. Set up a loop to repeatedly call the processing function rzrqpk until the parameter PercentDone = 100.

```
EXPORT32 long FAR EXPORT rzrqpk(float *x, float *y, long imax,
float *shape, long numshape,
float *yout, long numout,
char *WorkingSpace, long size_workspace,
float *PercentDone, long DelayTicks );
```

DelayTicks = number of CPU processor ticks allowed before the RazorQPK DLL yields control of the processor.

The RazorQPK DLL is Windows-friendly. It processes data for the allowed number of clock ticks, and then returns control to the calling program. When the calling program is ready, it may send a signal to the DLL to process a little more. This back-and-forth loop continues until the DLL sends a signal that the processing is complete and the results are ready.

5. Recover any desired arrays or parameters from the workspace.

6. Delete the workspace.

Storage Arrays in the QPK workspace

Ithings Array

The Ithings array stores (long) integer variables at the beginning of the workspace used by RazorQPK. The array has 100 locations, filled as described below. (The parameters saved in reserved locations are not guaranteed to be the same in all versions of RazorQPK. The parameters below that are shown in **bold type and underlined** are maintained in all versions of the RazorQPK DLL.)

0 Requested number of iterations.

- When the workspace is initialized by the function InitWorkSpaceQpk, this location is filled with the number 1.

1 Peak Picker # iperf. Only -4,-3,-1,0,1,2,3,4,5,10,11,12,13 are acceptable
2 Reserved. Baseline type. (0=Offset. -1=None. 2=Same shape as envelope)
3 Completed number of iterations.
4 Noise statistics istat. Only 0 and 1 are permitted.
5 Reserved. ifirst = index where x_value>adductionmass)
6 Reserved. Not used.
7 Reserved. baseline flag (0 = yes, -1=no, 1=envelope shape)
8 Reserved. bzero
9 Reserved. jmxmx
10 Reserved jmin
11 Reserved. jmax
12 Reserved. lpmin = processing start index in image
13 Reserved. ipmax = processing end index in image
14 Number of peaks in peak table.
15 Clock ticks used.
16 State vector. Current subroutine
17 State vector. Current position within subroutine
18 State vector. Current j index
19 State vector. Current I index
20 State vector. Current k index
21 Reserved. Saves parameters during Station Breaks (ileft, kto)
22 Reserved. Saves parameters during Station Breaks (iright, kfar)
23 Reserved. Saves parameters during Station Breaks (left point of current processing region)
24 Reserved. Saves parameters during Station Breaks (right point of current processing region)
25 Reserved. imax
26 Reserved. gsize
27 Reserved. npomax

28 Reserved. longkmax
29 Reserved. kmax
30 Reserved. ntimag, jmin in rzkern
31 Reserved. jemin, jmax in rzkern
32 Reserved. jemax
33 Reserved. lekmin
34 Reserved. lekmax
35 Reserved. isamp
36 Maxpks_fit – maximum # peaks allowed in a setup for rzrfit.
37 Reserved. For qpeaks2, peakshape_type.
38 Reserved. For qpeaks2, fix_width.
39 Reserved. For qpeaks2, fix_fourthparm.
40 Reserved. For qpeaks2, fix_fifthparm.
41 Reserved. For qpeaks2, tiny_peakshape_type.
42 Reserved. For qpeaks2, group_shape_linker.
43 Reserved. For qpeaks2, max_groupsize.
44 Reserved.
45 Reserved.
46 Reserved.
47 Reserved.
48 Reserved.
49 Reserved.
50 Size of ithings.
51 Size of fthings
52 Reserved. Size1
53 Reserved. Size2
54 Reserved. Size3
55 Reserved. Size4
56 Reserved. Size5
57 Reserved. Size6
58 Reserved. Size7
59 Reserved. Size8
60 Reserved.
61 Location (offset) of fthings in workspace
62 Location (offset) of work1 in workspace = baseline array
63 Location (offset) of work2 in workspace
64 Location (offset) of work3 in workspace = smoothed data array
65 Location (offset) of work4 in workspace
66 Location (offset) of work5 in workspace
67 Location (offset) of work6 in workspace
68 Location (offset) of work7 in workspace
69 Location (offset) of work8 in workspace = peak table
70 Reserved.
71 Reserved.
72 Reserved.
73 Reserved.

74 Reserved.
75 Reserved.
76 Reserved.
77 Reserved.
78 Reserved.
79 Reserved.
80 Reserved.
81 Reserved.
82 Reserved.
83 Reserved.
84 Reserved.
85 Reserved.
86 Reserved.
87 Reserved.
88 Reserved.
89 Reserved. Temporary value of npks
90 Reserved.
91 Reserved.
92 Reserved.
93 Reserved.
94 Reserved.
95 Reserved.
96 Reserved.
97 Reserved.
98 Reserved.
99 Reserved.

Fthings Array

The fthings array stores float (double) variables in the workspace used by RazorQPK (RazorDQPK).

Fthings has 50 - 100 locations (depending on version of RazorQPK). The size of fthings (number of locations) is stored in ithings[51]. The location of the start of the fthings vector is stored in ithings[61].

Fthings is filled as described below. The parameters saved in reserved locations are not guaranteed to be the same in all versions of RazorQPK. The parameters below that are shown in **bold type and underlined** are maintained in all versions of the RazorQPK DLL.

0 sigma
1 poiscns
2 basewidthmult
3 singlet width (The units are amu if singletwidth > 0. If singletwidth < 0, then the value = Resolution or Resolving power.)
4 Reserved.
5 signal2noise
6 Reserved. Current sigma
7 Reserved. Current poiscns
8 Reserved.
9 Reserved.
10 Reserved.
11 Reserved.
12 Reserved. specwidth = spectrometer width in bins
13 Reserved. delo
14 Reserved. bzero (noninteger)
15 isotopicwidth = isotopic width in Da
16 Reserved. Max vaue in image
17 Reserved. Mean counts in image
18 Reserved. Saves parameters during Station Breaks
19 Reserved. Saves parameters during Station Breaks
20 Reserved. Saves parameters during Station Breaks
21 Reserved. Saves parameters during Station Breaks
22 Reserved. Saves parameters during Station Breaks
23 Reserved. Saves parameters during Station Breaks
24 Reserved. Saves parameters during Station Breaks
25 Reserved. Saves parameters during Station Breaks
26 Reserved. Saves parameters during Station Breaks
27 Reserved. Saves parameters during Station Breaks
28 Reserved. Saves parameters during Station Breaks
29 Reserved. Saves parameters during Station Breaks
30 Reserved.

31 Reserved.
32 Reserved.
33 Reserved.
34 Reserved.
35 Reserved. For qpeaks2, min_width.
36 Reserved. For qpeaks2, max_width.
37 Reserved. For qpeaks2, fourthparm.
38 Reserved. For qpeaks2, min_fourthparm.
39 Reserved. For qpeaks2, max_fourthparm.
40 Reserved. For qpeaks2, fifthparm.
41 Reserved. For qpeaks2, min_fifthparm.
42 Reserved. For qpeaks2, max_fifthparm.
43 Reserved. For qpeaks2, full_fit_range.
44 Reserved.
45 Reserved.
46 Reserved.
47 Reserved.
48 Reserved.
49 Reserved.
50 Reserved.

QPKDemo

QPKDemo is furnished as
(a) **RazorQPK.DLL**, the DLL that contains the required processing functions. (Under special arrangement, Qpeaks can be provided for operating systems other than Windows.)
(b) **QPKdemo.cpp**, source code for a demonstration console program that reads an XY data file, receives input from the user, allocates workspace, and writes output files.
(c) **Qpkqpk.cpp**, source code for the functions that call the RazorQPK DLL and control the processing.
(d) **Mopread.cpp,** source code for functions to read/write ASCII XY data files.
(e) **QPKdirect.cpp**, source code for a demonstration console program that reads an XY data file, receives input from the user, allocates workspace, and writes output files. QPKdirect calls the processing functions within the RazorQPK DLL directly, rather than using the intermediate functions of qpkqpk.cpp.

Overview: QPKDemo Processing Functions

When using QPKDemo, the programmer needs to fill the input structure, then call the following 3 functions (which are presented as source code in qpkqpk.cpp): These functions provide calls into the RazorQPK DLL.

- SetupNewTaskQpk, calculates size of workspace needed. Also calculates size of output array.

  (Programmer then allocates a workspace array, and an output array.)

- SetupWorkSpaceQpk, initializes the workspace.

- PerformTaskQpk, interacts with the RazorQPK DLL and controls the processing loop. At the end of processing, the output array will be filled, and a table of peak positions, heights, widths, etc. will be available within the workspace.

The following structure, defined in the file mopdemo.h, contains the input parameters needed by the RazorQPK DLL for QPK processing.

```
struct qpk_setup_struct
// This structure combines all float and long values, and all array pointers,
//      that a user (programmer) must specify for the RazorQPK peak picking engine:
{
        float   singlet_width;          // Required.
                                        // Spectrometer (damage) width in data,
                                        // If singlet_width is a POSITIVE number, it is
                                        // interpreted as a constant width in the input data.
                                        // If singlet_width is a NEGATIVE number, it is
                                        // interpreted as constant resolution (m/z)/(delta_m/z)

        float   peak_signal2noise;      // Required.
                                        // Signal to noise criterion for picking peaks
                                        // Use positive peak_signal2noise for height criteria
                                        // Use negative peak_signal2noise for area criteria
                                        // Note: peak_signal2noise values -1 to –5 (area criteria)
                                        // seem to work well for mass spec data.

        long    picker;                 // Optional.  May be set to 0.
                                        // Same as iperf in razor manual.
                                        // Acceptable values = -1,0,1,2,3,4,5,10
                                        // Recommended picker = 10.
                                        // If picker = 0, picker = 10 will be used.

        long    noise_statistics;       // Optional.  May be set to 0.
                                        // Same as istat in razor manual.
                                        // Acceptable values = 1 (gaussian/normal noise)
                                        // or 2 (poisson noise)
                                        // If noise_statistics = 0.0, gaussian/normal will be used.

        float   rms_noise;              // Optional.  May be set to 0.
                                        // Rms noise (if known).
                                        // Used when noise_statistics = 1
                                        // Input 0.0 if not sure.

        float   poisson_scans;          // Optional.  May be set to 0.
                                        // # scans which have been averaged in current data.
                                        // Effectively rescales data to # counts.
                                        // Used when noise_statistics = 2
                                        // Input 0.0 if not sure.

        float   baseline_width_multiplier;  // Used if need a rzredg baseline in the problem.
                                        // Adding a rzredg baseline is recommended.
                                        // Recommend using multiplier 3 - 10.
                                        // If this parameter is zero, a baseline will not be used.

        float *input_masses;            // pointer to spectrum masses array (required).
        float *input_intensities;       // pointer to spectrum intensities array (required).
        long    num_datapoints;         // size of xdata, ydata arrays
                                        // require that num_datapoints > zero.

        // NOTE: The following input spectrometer shape function is
```

```
        // not yet implemented in the RazorQPK DLL!
        // Therefore, these inputs are ignored at present.
        float *input_spectrometer_shape_func;  // pointer to input spectrometer shape array
                                               // (may be NULL).
        long    num_shapepoints;               // size of spectrometer shape function array
                                               // if num_shapepoints = 0, spectrometer shape is
                                               // assumed gaussian, with width = singlet_width
};
```

The 3 processing control functions that drive QPK processing, by calling functions in the RazorQPK DLL, also use the following information:

- **DelayTicks** = number of CPU processor ticks allowed before the RazorQPK DLL yields control of the processor. The RazorQPK DLL is windows-friendly. It processes data for the allowed number of clock ticks, then returns control to the calling program. When the calling program is ready, it may send a signal to the DLL to process another round. This back-and-forth loop continues until the DLL sends a signal that the processing is complete and the results are ready.
- **quiet =** flag that allows printf during execution.

Detail: Programming calls for QPeak

The steps for running Qpeaks from QPKDemo are as follows:

1. Fill the qpeak input structure.

2. Get the required size of the workspace array, and the required size of the output (result) array by calling SetupNewTaskQpk.

```
long SetupNewTaskQpk(struct qpk_setup_struct *qpksetptr,
    unsigned long *sizeWorkSpace, long *numOutArray,
    long quiet );

    // input qpksetptr
    // output sizeWorkSpace
    // output numOutArray
    // Use quiet = 0 unless debugging
```

3. Allocate the workspace and the output array.

4. Call SetupWorkSpaceQpk to initialize the workspace.

```
long SetupWorkSpaceQpk(struct qpk_setup_struct *qpksetptr,
    char *WorkSpace, unsigned long sizeWorkSpace,
    float *outputYArray, long numOutArray,
    long quiet);

    // input qpksetptr
    // input array WorkSpace, size sizeWorkWpace
    // input array outputYArray, size numOutArray
    // Use quiet = 0 unless debugging
```

5. Run PerformTaskQpk.

```
long PerformTaskQpk(struct qpk_setup_struct *qpksetptr,
    char *WorkSpace, long sizeWorkSpace,
    float *outputYArray, long numOutArray,
    long delayTicks, long *completedIters,
    float *percentDone, long quiet);

    // input qpksetptr
    // input array WorkSpace, size sizeWorkWpace
    // input array outputYArray, size numOutArray
    //       outputYArray will be filled during processing
    // input delayTicks = time allowed for processing before control returned to caller
    //       delayTicks enables programmer to maintain control
    //       and prevent long processing delays under Windows
    // output completedIters
    // output percentDone
    // Use quiet = 0 unless debugging
```

6. Recover the peak table from the workspace.

   The RazorQPK workspace begins with an integer (long) array that contains information on (a) the state of the processing and (b) the locations of all arrays used in the processing. *The offset of the peak table from the beginning of the workspace is stored in the 70th (long) integer location of the workspace*. *The number of peaks in the peak table is stored in the 15th (long) integer location of the workspace*.

   The following example code, from qpkqpk.cpp, accesses the table.

```
// write out the final peak table
if( *percentDone == 100.0F ){
   // offset of table from beginning of workspace is in glthings[69];
   // number of peaks in the table = glthings[14];
   gTableResults  = (float *)(WorkSpace+glthings[69]);
   iwrite_err = tablesave("TABLEQF.DAT", gTableResults, glthings[14], 0L);
   printf( "Wrote file TABLEQF.DAT  # Peaks = %ld \n", glthings[14] );
}
// end write out final peaktable
```

7. Delete the workspace.