INEL/EXT-97-00275

April 1997

Gauss Algorithms User Manual

Ann E. Egger

Note: If you are viewing this manual online with Acrobat, click on the second icon from the left in the toolbar.This will show the bookmarks, which will help you navigate in this document.

INEL/EXT-97-00275

Gauss Algorithms User Manual

Ann E. Egger

Published April 1997

Idaho National Engineering Laboratory
National Security Programs
Lockheed Martin Idaho Technologies Company
Idaho Falls, Idaho 83415

Prepared for the
U.S. Department of Energy
through the INEL LDRD Program
Under DOE Idaho Operations Office
Contract DE-AC07-94ID13223

SUMMARY

The Gauss Algorithms have been developed for use in analyzing gamma-ray spectra and are available as a library of compiled functions. They can be used to develop spectroscopy systems for nuclear research, waste management, national security, and other application areas.

This manual describes for the user:

- what each function does
- how to prepare the input for each function
- how to invoke each function
- and how to interpret the output from each function.

It also indicates how to acquire a copy of the Gauss Algorithms library, where to locate example code, and how to get more information.

ACKNOWLEDGMENTS

Many people have contributed to the to the development of the Gauss Algorithms library. Special thanks goes to the following individuals:

To A. J. (Gus) Caffrey for locating funds for the development of the Gauss Algorithms library.

To Harry Sauerwein and Myra D. Anderson for locating funds for the documentation and in-house release of the Gauss Algorithms library.

To Marie H. Putnam for extracting the essence of the algorithms from Gauss VII and producing usable FORTRAN subroutines. The Gauss Algorithms library began as a reimplementation in the C programming language of those FORTRAN subroutines.

To Richard G. Helmer for answering many questions about the GAUSS VII algorithms and about the history of GAUSS.

To John W. Mandler for providing the In-Plant Source Term references.

To Ken M. Krebs for help in compiling and testing the Gauss Algorithms library on the PC platform, and for being the first to put the Gauss Algorithms library to use implementing a special purpose system.

To James C. Byers for designing the content and style of the User Manual with the user always in mind.

To Kurt Welker for using his software metrics to demonstrate the need to reimplement the FORTRAN routines and for reviewing the design of the User Manual and software maintenance manual.

To William E. May for editing the User Manual and creating the web page used to distribute the Gauss Algorithms Library within the INEEL.

To David Combs for developing the graphics used in the web page.

CONTENTS

Figures

Tables

Gauss Algorithms User Manual

1. GENERAL DESCRIPTION OF GAUSS ALGORITHMS

The Gauss Algorithms have been developed to aid in spectral analysis. They have been used in a series of nine spectral analysis programs named GAUSS I through GAUSS IX. The basic purpose of the GAUSS programs is to analyze the peaks in a γ-ray spectrum. This involves determining the channel position of a peak, which can be converted into the γ-ray energy of the peak, and determining the peak area, which can be converted into the number of γ-rays of that energy emitted by the source. The fit to the peak data is carried out by a nonlinear least squares fitting routine to determine the parameters (height, width, and position) of a Gaussian function that represents the data.

Thus, one of the uses of the Gauss Algorithms library has been to implement special-purpose products for analyzing γ-ray spectra from Ge semiconductor detectors. These routines provide the ability to calibrate energy, calibrate peakwidth, search for peaks, search for regions, and fit the spectral data in specifiable regions of γ-ray spectra. A spectral data fit of a region is the sum of one or more Gaussian components. With Gauss Algorithms, a user can determine the coordinates not only to plot the curve of this fit, but to plot the curve of each Gaussian component as well.

The calibration of energy and peakwidth use Bevington's Gauss-Jordan elimination to invert a symmetric matrix and calculate its determinant as part of a linear least squares fit. The peak search determines the initial location of peaks with the crossproduct of the spectrum with the unit squarewave. Then it uses a linear least squares fit to refine the location. The region search computes a background curve and from that, locates likely regions to be fitted. The spectral data fit uses the Levenberg-Marquardt algorithm as implemented in Minpack's lmder routine. Minpack is by Jorge More', Burt Garbow, and Ken Hillstrom at Argonne National Laboratory and includes software for solving nonlinear equations and nonlinear least squares problems; it is available via NETLIB at http://www.netlib.org/liblist.html. A square symmetric covariance matrix is calculated from the fit result, and fit uncertainties are calculated from the matrix.

For purposes of analyzing with Gauss Algorithms, a spectrum is expressed in terms of channels and counts. The horizontal axis of the graph represents the channels, and the channel range can be any range of integers. This range of channels is closely related to energy. Each channel represents an energy range, and energy increases with channels. Gauss Algorithms uses an energy equation to relate channels to energy. The vertical axis of the spectral graph represents the count of gamma-rays emitted by the source at an

energy range that corresponds to a channel. See Figures 1a and 1b for examples of a spectrum drawn discretely and in a Manhattan skyline style.

A "peak fitting region" or "region" is a range of channels within the spectrum. Analysis or fitting of the spectral data is done over one region at a time. See Figure 2 for illustrations of a region of a spectrum.

A "peak centroid" or "peak" is a channel within a region chosen by the user or the fitting algorithm that indicates the location of a Gaussian component. See Figure 3 for illustrations of a peak in the spectrum.

One of the functions in the Gauss Algorithms suite fits a region using any user-chosen peaks and the spectrum as input. It returns information that includes a fit of each Gaussian component, the sum of all the component fits, called a "region fit," and a "background" of the region fit. See Figure 4 for illustrations of Gaussian components, a region fit, and its background.

Whenever the energy or peakwidth are calibrated, a "reduced chi squared" (χ^2_R) of the calibration is calculated. χ^2_R is also calculated for every fit of a region. A small χ^2_R value indicates a good corresponding calibration or fit.

The Gauss Algorithms routines are independent. Any one or more of them can be used in an application. They are written in ANSI C and can be used on any computer with an ANSI C compiler. The Levenberg-Marquardt algorithm from Minpack is available in either C or FORTRAN. The FORTRAN version came from NETLIB and the C version came from using the *f2c* filter on the FORTRAN version. The library that is released will normally be compiled with the C version of the Levenberg-Marquardt algorithm.

The Gauss Algorithms require minimal memory and on a 150-Mhz Silicon Graphics Indigo system, calibration, searches, and postprocessing all take hundredths of a second to complete (both computer time and real time). A typical search is for hundreds of peaks and regions spread over 8,000 channels. Fitting a region takes longer. One 150-channel region, no peaks, one cycle, takes about 0.1 seconds. No peaks, 5 cycles, takes about 0.7 seconds. No peaks, 9 cycles, takes about 2 seconds. 5 peaks, one cycle, takes about 0.25 seconds. Table 1 shows these times.

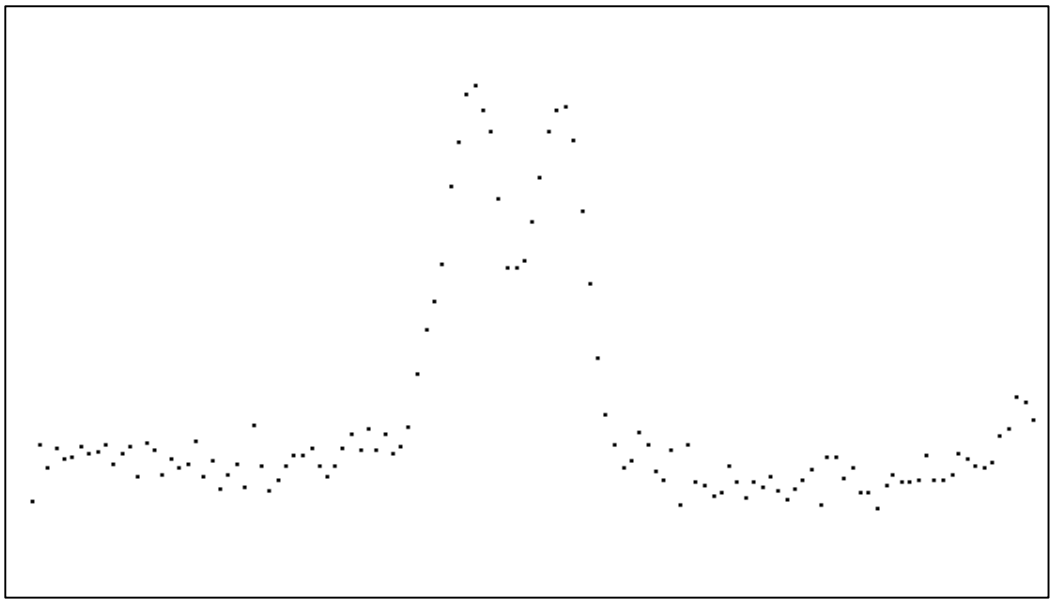

Figure 1a. Spectrum (discrete format).

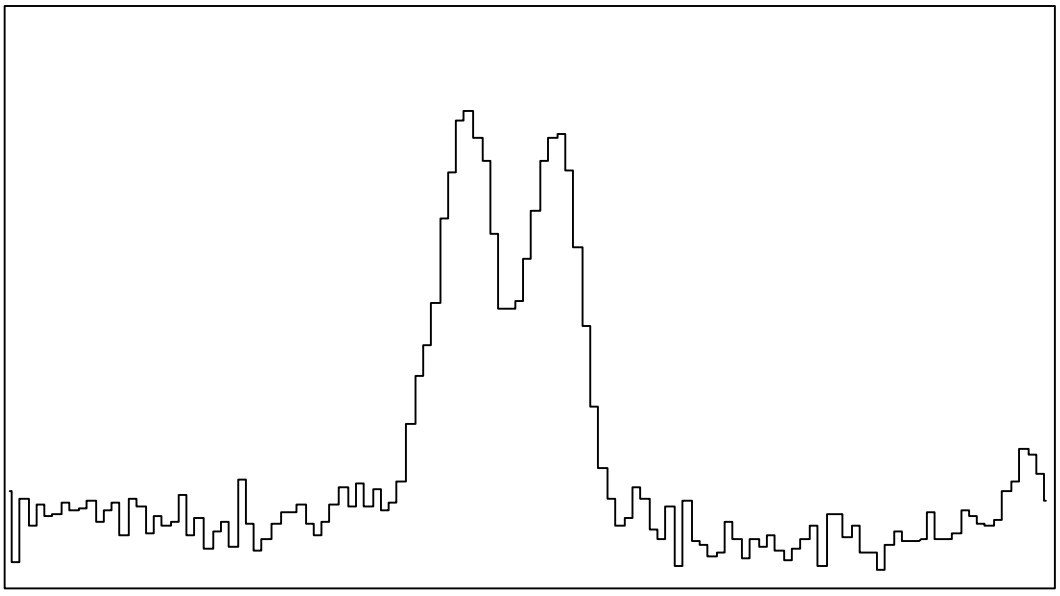

Figure 1b. Spectrum (Manhattan Skyline format).

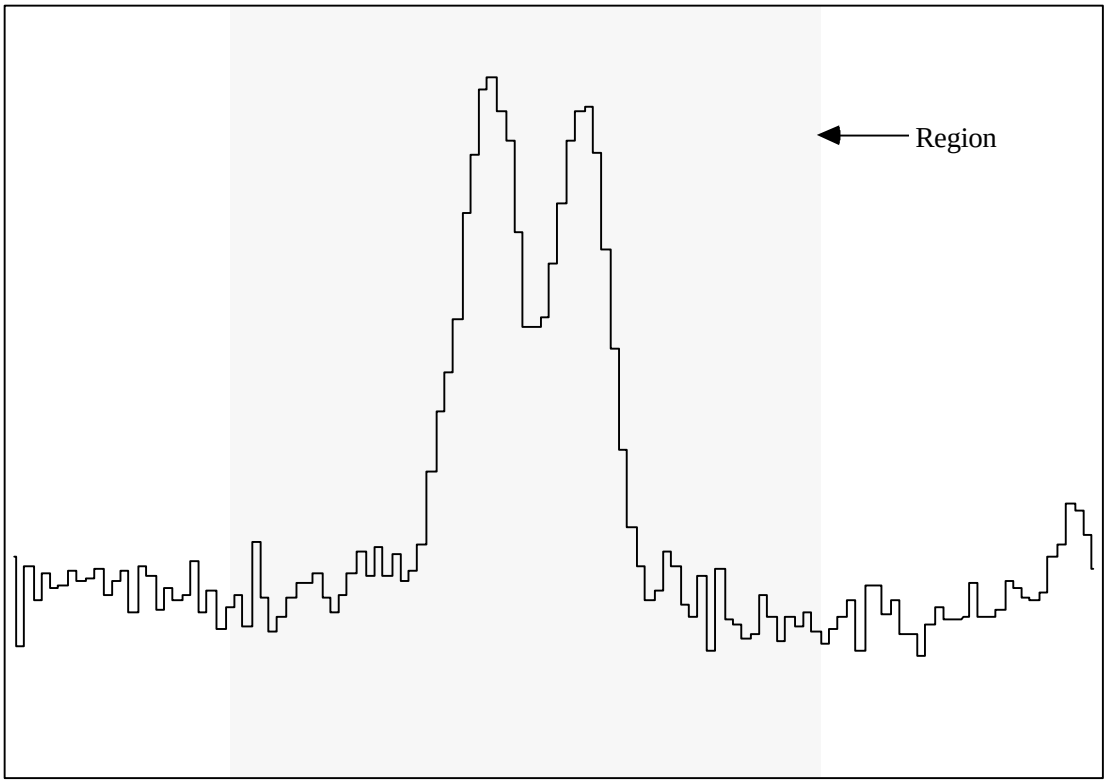

Figure 2. A “peak fitting region” or “region” is a range of channels within a spectrum.

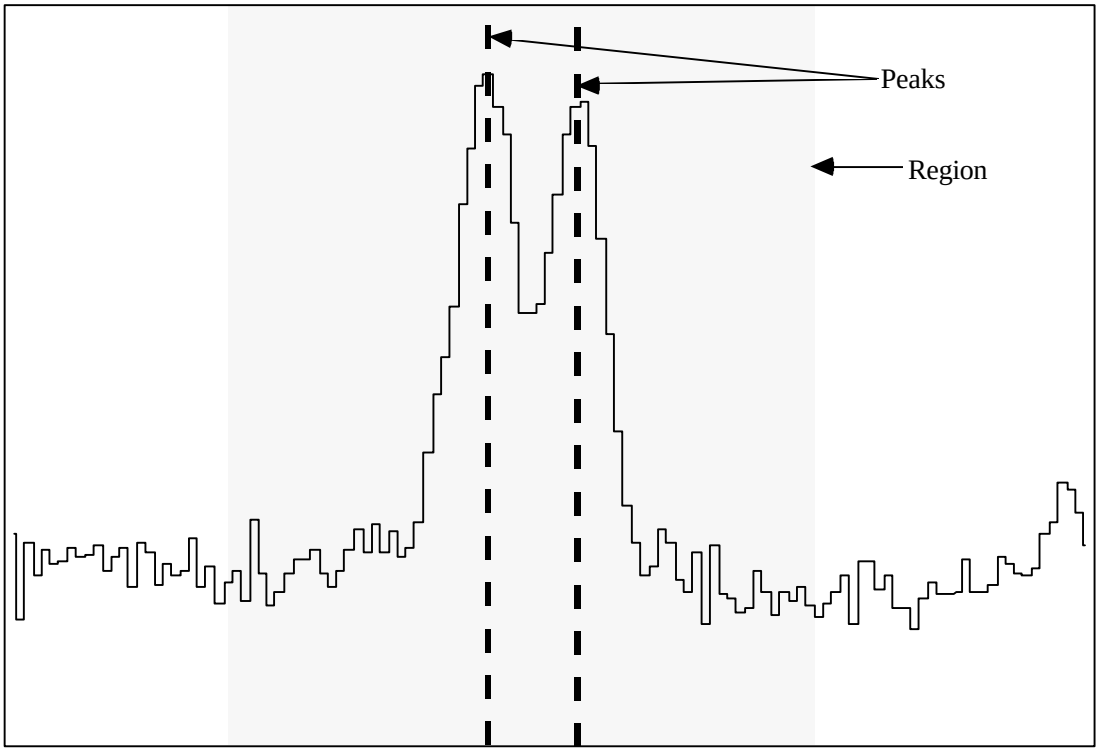

Figure 3. "Peak centroids" or "peaks" are specific channels within a chosen region, indicating the location of a Gaussian component.

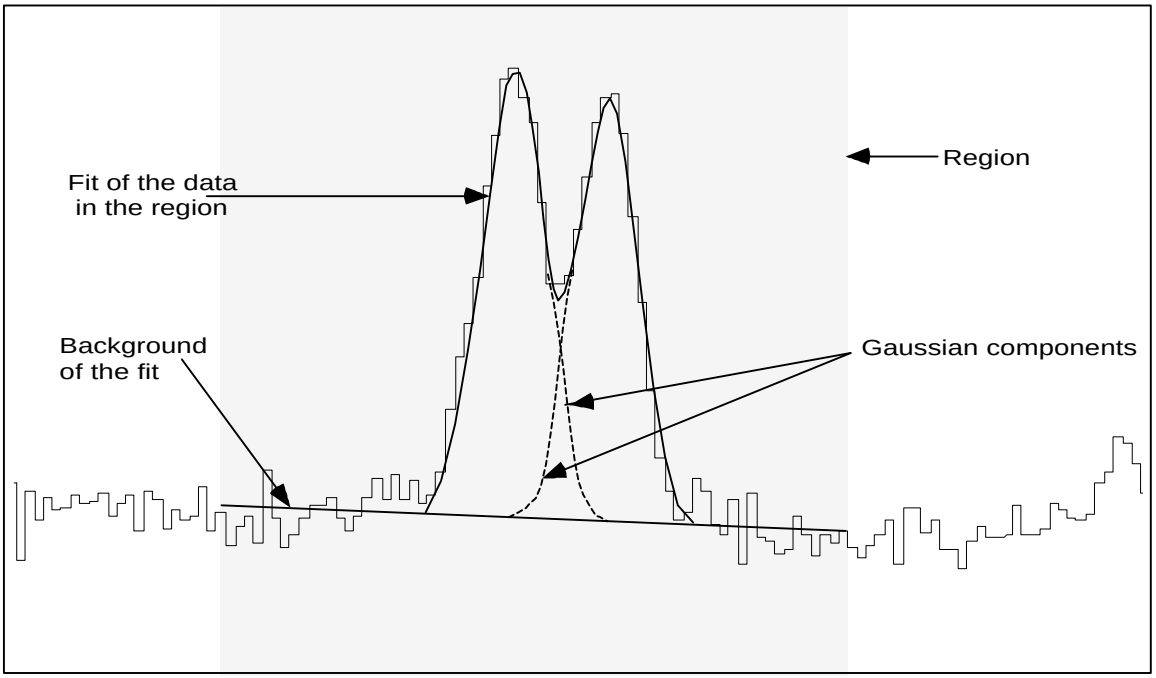

Figure 4. Gaussian components, a region fit, and its background.

Table 1. Times to fit one 150-channel region.

	1 cycle	5 cycles	9 cycles
0 peaks	.1 sec	.7 sec	2 sec
5 peaks	.25 sec	-	-

2. HISTORY OF THE GAUSS PROGRAMS AND ALGORITHMS

2.1 Purpose of the GAUSS Programs

The basic purpose of the GAUSS programs is to analyze the peaks in a γ-ray spectrum. This involves determining the channel position of a peak, which can be converted into the γ-ray energy of the peak, and the peak area, which can be converted into the number of γ-rays of that energy emitted by the source. The fit to the peak data is carried out by a nonlinear least squares fitting routine to determine the parameters (i.e., height, width, and position) of the Gaussian function that represents the data. The programs were generally designed to carry out the spectral analysis (1) in an automated way, that is, with a minimum of user input, and (2) in a manual way, with user input for each peak. The availability of two levels of user interaction has allowed the GAUSS programs to be used both in the production mode to get results from many spectra, and in the metrology or research mode to get the very best results for each peak of interest. The success of the code for the metrology mode is illustrated by a long list of publications of measured γ-ray energies (or energy differences) (see References, under "Gamma-Ray Energy Measurement References"). It has also been used for precise γ-ray intensity measurements (see References, under "Gamma-Ray Intensity Measurement References"). The success in the production mode is indicated by the development of special versions discussed below and their use for in-plant source term measurement studies (see References, under "In-Plant Source Term Measurement Studies").

2.2 Versions of the GAUSS Program

Various versions of the GAUSS code have been developed since the early 1960s, and some have been documented (see References, under "Gauss-Related References"). Several versions of the GAUSS code have been used extensively. Table 2 lists the versions of the code and some characteristics of each version. Following Table 2 are further explanatory notes, differentiating the versions.

Table 2. GAUSS Program Versions.

Version	Type	Derived from	Language	Machine	Input Style	Peak Width Equa-tion	Automatic Peak Locating	Automatic Region Locating	Radionu-clide Series
GAUSS I	research	—	FORTRAN IV	IBM 7040	batch	no	no	no	no
GAUSS II	develop-mental[a]	—	—	—	—	—	—	—	—
GAUSS III	research	GAUSS II	FORTRAN IV	IBM 7040	batch	yes	no	no	no
GAUSS IV	develop-mental[a]	—	—	—	—	—	—	—	—
GAUSS V	research	GAUSS IV	FORTRAN IV	IBM 360/75	batch	yes	yes	no	no
GAUSS VI	produc-tion	GAUSS V	FORTRAN IV	IBM 360/75	batch	yes	yes	no	yes
GAUSS VII	research	GAUSS V	FORTRAN IV	CDC CYBER-176	batch	yes	yes	yes	no
GAUSS VIII	produc-tion	GAUSS VI	FORTRAN V	CDC CYBER-176	batch	yes	yes	yes	yes
GAUSS IX	research	GAUSS VII	C	Silicon Graphics IRIX	inter-active	yes	yes	yes	no
Gauss Algorithms	any	GAUSS VII	C and FORTRAN 77	any	batch	yes	yes	yes	no

a. Documentation is not available on GAUSS II and GAUSS IV code.

GAUSS I was a research version based on the experience with NaI(Tl) detectors. This shows up in the non-Gaussian portion of the peak shape, which had symmetric tailing on both sides of the main Gaussian peak. The program was then applied to spectra that were obtained from the early Ge(Li) detectors. The incentive for reporting on the program may have come from the Ge(Li) spectral analysis, because the Idaho National Engineering Laboratory's (INEL's) prime method for analyzing NaI(Tl) spectra was based on the linear least squares fitting of the whole spectrum with components representing either single γ-rays or single nuclides. The GAUSS I program was small enough to allow the complete listing to be given in the associated report.

GAUSS II and GAUSS IV were developmental versions that were not extensively used or documented.

GAUSS III was the next research version and was an expansion and tailoring of the earlier version directed more toward the analysis of spectra from Ge(Li) detectors. The program included the capability to use a selected set of peaks in the spectrum to determine the γ-ray energy as a function of the channel. Also, a peak width function could be used to provide the initial width values. The spectral background under the peaks was a linear function. The program would fit up to five Gaussian components in a single portion of the spectrum. Each of the corresponding 30 parameters could be allowed to vary in the nonlinear least squares fit, or each could be fixed at its initial value. The peaks could be fit as pure Gaussian functions, or tailing could be added. However, the tailing function was symmetric and was from INEL's earlier work with Nal(Tl) detectors. The report on this version included a discussion of the precision of the results that could be obtained, especially for the γ-ray energies. Since the peaks from Ge(Li) or Ge detectors have tailing on the low-energy side, for energy determinations it was common practice to only use the data near the top of the peak (e.g., from the half-height on the low-energy side to just a little lower on the high-energy side). This was not necessarily the best option for determining the peak area, but it avoided the low-side tailing. This choice also meant that one could not achieve consistency between fits to singlets and doublets since for the doublets one would have to fit all the way down on the high side of one peak and the low side of the other.

GAUSS V was the next research version, which expanded the capability to treat spectra from Ge(Li) and Ge detectors. Since these spectra often have over a hundred peaks, a routine was added to automatically find peaks. This was done by means of a correlation method that reinforced the peak structure and subtracted a continuous background, ideally leaving just the peaks. These results were then used as the starting positions for fitting the original data with a Gaussian function. As before, a single fit could treat up to five Gaussians on a linear background. Four different levels of input

were available, so at one extreme the user could let the program find the peaks and analyze them with only a minimum of input related to the whole spectrum, or at the other extreme the user could define the initial parameters for the fit to each peak. The program could use some peaks in the spectrum to determine the energy channel relationship. In this energy calibration process, a previously determined correction for the nonlinearity of the electronics could be used, after which the energies of all of the peaks were determined. Similarly, the full-energy peak efficiency could be previously stored in a data file and used to compute the relative emission rate for each peak. This program was made generally available through the Radiation Shielding Information Center (RSIC) at Oak Ridge National Laboratory (ORNL).

GAUSS VI was the first version that was designed for production use. The main addition to GAUSS V was the use of a library of γ-ray energies and intensities for the radionuclides of interest. The energies of the γ-ray peaks were then compared to the energies of the lines in the library to determine which radionuclides might be present in the sample; decay rates of the radionuclides were determine from the peak area, detector efficiency, and the intensity from the library. For nuclides with more than one observed γ-ray, the decay rates from the various γ-rays were compared to test the correctness of the nuclide identification and when they were consistent, they were averaged to get the final value. The program was also modified to increase the chance that an analysis of each peak would be completed without a failure. The detailed analysis of each peak that was available in GAUSS V was eliminated. This program was also made available through the RSIC library at ORNL.

GAUSS VII version was the next research version and as such was based on GAUSS V, and did not include the nuclide identification portions of GAUSS VI. The development goal was primarily to incorporate as automatic features some of the operations that the GAUSS V users commonly employed to identify and fit multiplets. A common problem was that the earlier peak location routine would only identify one peak in a region that clearly contained two or more peaks. The user then had to identify the additional peaks and extend the range of data used to satisfactorily define the spectral background under the peaks. The first step in addressing this problem was to add a routine to determine peak regions. This was done by successive smoothings of a spectrum until it became a smooth function that closely represented the spectral background under the peaks. The regions of the actual spectrum that were significantly above this background spectrum were taken to be made up of peaks, even if the peak location routine had not identified peaks in the region. Even where peaks had been identified, these peak regions could be used to define the range of data that included other weaker peaks and thus define the region to include in a peak fit. The fit of a peak region then began with all of the peaks identified by the peak location routine. A second addition

allowed the program to automatically add peak components in any fitting region where the initial fit did not satisfactorily fit the measured data. For any fit region, up to five new peaks could be added, one at a time, with a maximum of ten peaks in any region. The user had options to (l) print out all of these fits or only the best one, (2) vary the statistical criterion used to determine whether a new peak should be added, (3) and define many other details of the fitting. This program was also made available through the RSIC library at ORNL.

GAUSS VIII was the second production program, that is, it was an extension of the GAUSS VI program. The main addition was to replace the radionuclide decay rate routine with one based on the concepts of Gunnink and Niday (1972). Instead of simply tallying the decay rate computed from each γ-ray associated with a nuclide, a least squares fit was done to obtain the decay rate. If all of the lines associated with a nuclide were only associated with that nuclide, this was simply a weighted average of the individual decay rates. However, in most complex spectra some of the lines were potentially associated with more than one nuclide. In this case, the least squares fit to the peak areas included all of the interrelated nuclides and their associated peaks. This method thus provided a composition of the complex peaks into the components from each contributing nuclide. To eliminate cases where a nuclide decay rate would be indeterminate (i.e., the fit would fail) and other complications, a great deal of data checking had to be programmed before the least squares fit to the peak areas.

GAUSS IX is the latest version of the GAUSS programs. It makes use of the computational routines of Gauss Algorithms in an interactive structure. The interactive features are implemented with OSF/Motif and the X Window System. This interactive version can dramatically decrease the turnaround time for spectral analyses, especially when the user needs to refit some of the peaks with specially-chosen fitting parameters. The graphic features increase the opportunity for detecting patterns and anomalies in the spectral analyses. Users of GAUSS IX can set up the analysis parameters (peaks, peak regions, etc.) interactively via window dialogs. They can interactively display and review the results, selectively re-fit peaks, and save or purge the results, as appropriate. The spectral displays include the spectral data, peak locations, peak fitting regions, fit curves, background curves, and each Gaussian component; and the display is completely zoomable, scrollable, and resizable.

3. SPECIFICS ABOUT GAUSS ALGORITHMS

3.1 Gauss Algorithm Glossary

Appendix A contains a glossary for the terms used in the Gauss Algorithms. Glossary items are listed alphabetically and are a useful reference for becoming familiar with the Gauss Algorithms.

3.2 Gauss Algorithm Functions

Appendix B alphabetically lists Gauss Algorithm functions. Each function is described by a definition, a description in words of the function's purpose/use, the return values of the function (as appropriate), and a list of other related functions and types. To use these functions in software, use the appropriate library and include file for your computer type, as shown in Table 3.

Table 3. Library and include file to use by computer type.

computer type	library	include file
Unix type	libgauss.a	GaussLib.h
Windows type	gauss.a	GaussLib.h

3.3 Gauss Algorithms Types

Appendix C alphabetically lists Gauss Algorithm types. Each type is described by a definition, a description in words of the type's purpose/use, and a list of other related functions and types. To use these functions in software, use the appropriate library and include file for your computer type, as shown in Table 3.

3.4 Reference to Electronic Copy of Example Code

Electronic copies of example code for the use of Gauss Algorithms functions and types is available from the Gauss Algorithms Home Page at http://home.inel.gov/gauss.

3.5 How to Get the Gauss Algorithms

INEL users can get the Gauss Algorithms from the Gauss Algorithms Home Page at http://home.inel.gov/gauss.

4. REFERENCES

Gamma-Ray Energy Measurement References

Greenwood, R. C., R. G. Helmer and R. J. Gehrke, 1970, "Precise Comparison and Measurement of Gamma-Ray Energies with a Ge(Li) Detector I 50-420 keV," *Nuclear Instruments and Methods* Vol. 77, pp. 141-158.

Greenwood, R. C., R. G. Helmer, and R. J. Gehrke, 1979, "Precise Gamma-Ray Energies for Calibration of Ge Semiconductor Spectrometers: up to 3.5 MeV," *Nuclear Instruments and Methods* Vol. 159, p. 465.

Helmer, R. G., 1979, "Gamma-Ray Energies for ^{228}Ra-^{228}Ac Decay and the ^{228}Th Decay Chain," *Nuclear Instruments and Methods,* Vol.164, p. 355.

Helmer, R. G., 1990, "Selected Gamma-ray Energies and Emission Probabilities for the Decay of ^{125}Sb and ^{154}Eu," *Applied Radiation and Isotopes*, Vol. 41, No. 1, pp. 75-81

Helmer, R. G., 1993, "Gamma-ray Energies for Calibration from the Decay of ^{161}Tb, ^{172}Hf-^{172}Lu, and ^{242}Am," *Nuclear Instruments and Methods*, Vol. A 330, p. 434, 1993.

Helmer, R. G., R. C. Greenwood, and R. J. Gehrke, 1971, "Precise Comparison and Measurement of Gamma-Ray Energies with a Ge(Li) Detector II 400-1300 keV," *Nuclear Instruments and Methods* Vol. 96, p. 173.

Helmer, R. G., R. J. Gehrke and R. C. Greenwood, 1975, "Peak Position Variation with Source Geometry in Ge(Li) Detector Spectra," *Nuclear Instruments and Methods* Vol. 123, p. 51.

Helmer, R. G., R. J. Gehrke, and R. C. Greenwood, 1979, "Gamma-Ray Energies for ^{40}K, ^{110m}Ag and the ^{226}Ra Decay Chain," *Nuclear Instruments and Methods*, Vol. 166, p. 547.

Helmer, R. G., A. J. Caffrey, R. J. Gehrke, and R. C. Greenwood, 1981, “Gamma-Ray Energies from the Decay of ^{99}Mo, ^{133}Ba and ^{210}Pb," *Nuclear Instruments and Methods*, Vol. 188, p. 671.

Helmer, R. G., and C. W. Reich, 1981, "Emission Probabilities and Energies of the Prominent Gamma-Ray Transitions from the Decay of ^{240}Pu," *Applied Radiation and Isotopes*, Vol. 32, pp. 829-831.

Helmer, R. G., C. W. Reich, R. J. Gehrke, and J. D. Baker, 1982, "Emission Probabilities and Energies of Gamma-Ray Transitions from the ^{239}Pu Decay," *Applied Radiation and Isotopes*, Vol. 33, pp. 23-26.

Helmer, R. G., and C. W. Reich, 1984, "Emission Probabilities and Energies of Gamma-Ray Transitions from the Decay of ^{235}U," *Applied Radiation and Isotopes*, Vol. 35, No. 8, pp. 783-786.

Helmer, R. G., and C. W. Reich, 1984, "Emission Probabilities and Energies of Gamma-Ray Transitions from the Decay of ^{238}Pu," *Applied Radiation and Isotopes*, Vol. 35, No. 11, pp. 1067-1069.

Helmer, R. G., and C. W. Reich, 1994, "An excited state of ^{229}Th at 3.5 eV," *Physical Review C, Nuclear Physics*, Vol. 49, p. 1845.

Reich, C. W., R. G. Helmer, J. D. Baker and R. J. Gehrke, 1984, "Emission Probabilities and Energies of Gamma-Ray Transitions from the Decay of ^{233}U," *Applied Radiation and Isotopes*, Vol. 35, No. 3, pp. 185-188.

Reich, C. W., and R. G. Helmer, 1990, "Energy Separation of the Doublet of Intrinsic States at the Ground State of ^{229}Th," *Physical Review Letters*, Vol. 64 , No. 3, p. 271.

Gamma-Ray Intensity Measurement References

Gehrke, R. J., R. G. Helmer, and R. C. 1977, Greenwood, "Precise Relative Gamma-Ray Intensities for Calibration of Ge Semiconductor Detectors," *Nuclear Instruments and Methods* Vol. 147, p. 405.

Helmer R. G., and C. W. Reich, 1984, "Emission Probabilities and Energies of Gamma-Ray Transitions from the Decay of ^{235}U," *Applied Radiation and Isotopes* Vol. 35, No. 8, pp. 783-786.

Helmer R. G., and C. W. Reich, 1985, "Emission Probabilities of Gamma-Ray Transitions Associated with the Decay of ^{241}Pu and ^{237}U," *Applied Radiation and Isotopes* Vol. 36, No. 2, pp. 117-121.

Helmer R. G., and C. W. Reich, 1984, "Emission Probabilities and Energies of Gamma-Ray Transitions from the Decay of ^{238}Pu," *Applied Radiation and Isotopes* Vol. 35, p. 1067.

Helmer, R. G. 1990, "Relative Gamma-ray Emission Probability for the Decay of ^{207}Bi," *Applied Radiation and Isotopes*, Vol. 41, No. 8, p. 791.

Helmer, R. G., and C. W. Reich, 1981, "Emission Probabilities and Energies of the Prominent Gamma-Ray Transitions from the Decay of ^{240}Pu," *Applied Radiation and Isotopes* Vol. 32, p. 829.

Helmer, R. G., C. W. Reich, R. J. Gehrke, and J. D. Baker, 1982, "Emission Probabilities and Energies of Gamma-Ray Transitions from the ^{239}Pu Decay," *Applied Radiation and Isotopes* Vol. 33, p. 23.

Helmer, R. G., 1983, *Precise Efficiency Calibration of Ge Semiconductor Detectors for 30-2800 keV Gamma-Rays,* EGG-PHYS-5735, November.

Reich, C. W., R. G. Helmer, J. D. Baker and R. J. Gehrke, 1984, "Emission Probabilities and Energies of Gamma-Ray Transitions from the Decay of ^{233}U," *Applied Radiation and Isotopes* Vol. 35, p. 185.

In-Plant Source Term Measurement Studies

Duce, S. W. et al., 1985, In-Plant Source Term Measurements at Brunswick Steam Electric Station, EGG-2392, NUREG/CR-4245, June.

Dyer, N. C. et al., 1978, In-Plant Source Term Measurements at Fort Calhoun Station - Unit 1, NUREG/CR-0140, July.

Dyer, N. C. et al., 1979, In-Plant Source Term Measurements at Zion Station, EG&G Idaho, Inc., NUREG/CR-0715, February.

Mandler, J. W. et al., 1980, In-Plant Source Term Measurements at Turkey Point Station - Units 3 and 4, NUREG/CR-1629, September.

Mandler, J. W. et al., 1981, In-Plant Source Term Measurements at Four PWR's, NUREG/CR-1992, August.

Mandler, J. W. et al., 1981, In-Plant Source Term Measurements at Rancho Seco Station, NUREG/CR-2348, October.

Mandler, J. W. et al., 1985, In-Plant Source Term Measurements at Prairie Island Nuclear Generating Station, EGG-2420, NUREG/CR-4397, October.

Gauss-Related References

Bevington, P. R., and D. K. Robinson, 1992, Data Reduction and Error Analysis for the Physical Sciences. New York: McGraw-Hill.

Cline, J. E., M. H. Putnam, and R. G. Helmer, 1973, *GAUSS VI, A Computer Program for the Automatic Batch Analysis of Gamma-Ray Spectra from Ge(Li) Spectrometers,* ANCR-1113.

Egger, A. E., M. H. Putnam, R. G. Helmer, A. J. Caffrey, and R. C. Greenwood, 1994, "GAUSS IX: An Interactive Program for the Analysis of Gamma-ray Spectra from Ge Semiconductor Detectors", IEEE Transactions on Nuclear Science, Vol. 42, No. 4, pp. 267-271.

Gunnink, R., and J. B. Niday, 1972, *Computerized Quantitative Analysis by Gamma-Ray Spectrometry*, Lawrence Livermore National Laboratory, UCRL-51061, Vol. I-IV.

Heath, R. L., R. G. Helmer, L. A. Schmittroth and G. A. Cazier, 1965, *The Calculation of Gamma-Ray Shapes for Sodium Iodide Scintillation Spectrometers, Computer Programs and Experimental Problems,* IDO-17017.

Helmer, R. G., R. L. Heath, L. A. Schmittroth, G. A. Jayne and L. M. Wagner, 1967, "Analysis of Gamma-Ray Spectra from NaI(Tl) and Ge(Li) Spectrometers, Computer Programs," *Nuclear Instruments and Methods*, Vol. 47, pp. 305-319.

Helmer, R. G., R. L. Heath, M. Putnam and D. H. Gipson, 1967, “Photopeak Analysis Program for Photon Energy and Intensity Determinations, Ge(Li) and NaI(Tl) Spectrometers," *Nuclear Instruments and Methods*, Vol. 57, pp. 46-57.

Helmer, R. G., and M. H. Putnam, 1972, *GAUSS V, A Computer Program for the Analysis of Gamma-Ray Spectra from Ge(Li) Spectrometers,* ANCR-1043.

Helmer R. G., and C. M. McCullagh, 1983, "GAUSS VII, A Computer Program for the Analysis of Gamma-Ray Spectra from Ge Semiconductor Spectrometers," *Nuclear Instruments and Methods* Vol. 206, p. 477.

Helmer, R. G., M. H. Putnam and C. M. McCullagh, 1986, "Nuclide Activities Determined from Gamma-Ray Spectra from Ge Detectors: A Review with GAUSS VIII as the Example," *Nuclear Instruments and Methods* Vol. A242, p. 42, invited talk at *Sixth Symposium on X- and Gamma-Ray Sources and Applications,* Ann Arbor, May 1985.

McCullagh, C. M., and R. G. Helmer, 1982, *GAUSS VII, A Computer Program for the Analysis of Gamma-Ray Spectra from Ge Semiconductor Spectrometers*, EGG-PHYS-5890.

Putnam, M. H., D. H. Gipson, R. G. Helmer and R. L. Heath, 1965, *A Nonlinear Least-Square Program for the Determination of Parameters of Photopeaks by the Use of a Modified Gaussian Function,* IDO-17016.

Putnam, M. H., R. G. Helmer and C. M. McCullagh, 1985, *GAUSS VIII, A Computer Program for the Nuclide Activity Analysis of Gamma-ray Spectra from Ge Semiconductor Spectrometers*, EGG-PBS-6768.

5. BIBLIOGRAPHY

Debertin, K. and Helmer, R.G., 1988. *Gamma- and X-Ray Spectrometry with Semiconductor Detectors*. Amsterdam: Elsevier.

Helmer, R.G., Cline, J.E., and Greenwood, R.C., 1975. "Gamma-ray Energy and Intensity Measurements with Ge(Li) Spectrometers" in *The Electromagnetic Interaction in Nuclear Spectroscopy*, W.D. Hamilton. ed., Amsterdam: North Holland Publishing Co. p. 775.

Helmer, R.G. and McCullagh, C. M., 1983. "GAUSS VII, A Computer Program for the Analysis of Gamma-ray Spectra from Ge Semiconductor Spectrometers. *Nuclear Instruments and Methods*, Vol. 206, p. 477.

Appendix A

Gauss Algorithms Glossary

GLCCType	type in GaussLib.h
GLChanRange	type in GaussLib.h
GLCurve	type in GaussLib.h
GLEfficiency	type in GaussLib.h
GLEgyEqnMode	type in GaussLib.h
GLEnergyEqn	type in GaussLib.h
GLEnergyPeaks	type in GaussLib.h
GLFitInfo	type in GaussLib.h
GLFitParms	type in GaussLib.h
GLFitRecList	type in GaussLib.h
GLFitRecord	type in GaussLib.h
GLPeakInfo	type in GaussLib.h
GLPeakType	type in GaussLib.h
GLPeaks	type in GaussLib.h
GLPkwdMode	type in GaussLib.h
GLPostInfo	type in GaussLib.h
GLRegions	type in GaussLib.h
GLRgnSrchMode	type in GaussLib.h

GLRtnCode	type in GaussLib.h
GLSpectrum	type in GaussLib.h
GLSplitMode	type in GaussLib.h
GLSummary	type in GaussLib.h
GLTypedPeaks	type in GaussLib.h
GLWidEqnMode	type in GaussLib.h
GLWidthEqn	type in GaussLib.h
GL_BADCALBLINF	possible value for GLRtnCode
GL_BADCHNRNG	possible value for GLRtnCode
GL_BADIRCH	possible value for GLRtnCode
GL_BADIRW	possible value for GLRtnCode
GL_BADMALLOC	possible value for GLRtnCode
GL_BADPEAKWD	possible value for GLRtnCode
GL_BADTHRESH	possible value for GLRtnCode
GL_CC_LARGER	possible value for GLCCType
GL_CC_LARGER_INC	possible value for GLCCType
GL_CC_SMALLER	possible value for GLCCType
GL_COVARR_DIM	definition used in GLFitRecord
GL_EGY_LINEAR	possible value for GLEgyEqnMode
GL_EGY_QUADRATIC	possible value for GLEgyEqnMode
GL_FAILURE	possible value for GLRtnCode

GL_FALSE	possible value for GLboolean
GL_MAX_EFFICY	definition used in GLEfficiency
GL_NUM_COEFFS	definition used in GLEfficiency
GL_OVRLMT	possible value for GLRtnCode
GL_PEAK_NORMAL	possible value for GLPeakType
GL_PEAK_RQD	possible value for GLPeakType
GL_PKWD_FIXED	possible value for GLPkwdMode
GL_PKWD_VARIES	possible value for GLPkwdMode
GL_RGNSRCH_ALL	possible value for GLRgnSrchMode
GL_RGNSRCH_FORPKS	possible value for GLRgnSrchMode
GL_SPLIT_ALLOWED	possible value for GLSplitMode
GL_SPLIT_NONE	possible value for GLSplitMode
GL_SUCCESS	possible value for GLRtnCode
GL_TRUE	possible value for GLboolean
GL_WID_LINEAR	possible value for GLWidEqnMode
GL_WID_SQRT	possible value for GLWidEqnMode
GL_chan_to_e	public function
GL_chan_to_w	public function
GL_curve	public function
GL_curve_alloc	public function
GL_curve_free	public function

GL_e_to_chan	public function
GL_ecalib	public function
GL_find_regnpks	public function
GL_fitrec_alloc	public function
GL_fitrec_free	public function
GL_fitregn	public function
GL_peaksearch	public function
GL_postprocess	public function
GL_prune_rqdpks	public function
GL_regnsearch	public function
GL_set_sigcount	public function
GL_spline	public function
GL_summ_alloc	public function
GL_summ_free	public function
GL_summarize	public function
GL_typed_peaks	public function
GL_wcalib	public function
GLboolean	type in GaussLib.h

Appendix B

Gauss Algorithms Functions

GL_chan_to_e

Definition:

```
void GL_chan_to_e(GLEnergyEqn    ex,
                  double         channel,
                  double         *energy);
```

Description:

GL_chan_to_e calculates the energy of the specified *channel* using the energy equation described in *ex*. The user must provide space for the answer in the *energy* parameter.

See Also:

GLEnergyEqn

GL_chan_to_w

Definition:

```
GLRtnCode GL_chan_to_w(GLWidthEqn    wx,
                       int           channel,
                       double        *width);
```

Description:

GL_chan_to_w calculates the peak width at the specified *channel* using the width equation described in *wx*. The user must provide space for the answer in the *width* parameter.

Return Values:

GL_SUCCESS, GL_FAILURE

See Also:

GLWidthEqn and GLRtnCode

GL_curve

Definition:

```
GLRtnCode GL_curve(GLSpectrum     spectrum,
                   GLFitRecord    *fit,
                   int            nplots_per_chan,
                   GLCurve        *curve);
```

Description:

GL_curve uses the information in *fit* to generate points to use in graphing the fit and background. GL_curve also uses the *spectrum* to calculate residuals.
The user can indicate the desired precision of the resulting curve with the *nplots_per_chan* parameter. *nplots_per_chan* = 1 would result in generating points only at each channel. *nplots_per_chan* = 2 would result in generating points midway between channels as well. The larger *nplots_per_chan* used, the more points will be generated between the channels.
The user must provide space for the answer in the *curve* parameter.

Return Values:

GL_OVRLMT, GL_SUCCESS

See Also:

GLSpectrum, GLFitRecord, GLCurve, and GLRtnCode

GL_curve_alloc

Definition:

```
GLCurve *GL_curve_alloc(int     regn_width,
                        int     nplots_per_chan,
                        int     npeaks);
```

Description:

GL_curve_alloc allocates memory for the GLCurve structure. The user should set *regn_width* to at least the number of channels in the region to be graphed, *nplots_per_chan* to at least the number of points to be graphed per channel, and *npeaks* to at least the number of peaks indicated in the GLFitRecord structure used by GL_curve later to set the contents of the GLCurve structure.

The value of GLCurve -> listlength is set properly in the returned structure.

Return Values:

If successful, GL_curve_alloc returns the address of the new GLCurve structure. If GL_curve_alloc fails, it returns NULL.

See Also:

GL_curve, GLCurve, GL_curve_free, GLFitRecord

GL_curve_free

Definition:

void GL_curve_free(GLCurve *curve);

Description:

GL_curve_free frees the memory of *curve* that was previously allocated by GL_curve_alloc.

See Also:

GLCurve, GL_curve_alloc

GL_ecalib

Definition:

```
GLRtnCode GL_ecalib(int          count,
                    double       *channel,
                    double       *energy,
                    double       *sige,
                    GLEgyEqnMode mode,
                    GLboolean    weighted,
                    GLEnergyEqn  *ex);
```

Description:

GL_ecalib uses channel-energy-uncertainty triplets to compute an energy equation. The number of triplets is indicated in the *count* parameter. The triplets are passed via the arrays *channel*, *energy*, and *sige*. The *mode* parameter indicates what kind of equation to produce. See GLEgyEqnMode for the current choices. The *weighted* parameter indicates whether to use the uncertainties in the *sige* array (weighted = GL_TRUE) or use the constant *1* for the uncertainties (weighted = GL_FALSE). The user must provide space for the answer in the *ex* parameter.

Return Values:

GL_BADMALLOC, GL_BADCALBLINF, GL_SUCCESS

See Also:

GLboolean, GLEnergyEqn and GLRtnCode, GLEgyEqnMode

GL_e_to_chan

Definition:

```
GLRtnCode GL_e_to_chan(GLEnergyEqn    ex,
                       double         energy,
                       double         *channel);
```

Description:

GL_e_to_chan uses the energy equation *ex* to compute the channel that corresponds to the given energy. The user must provide space for the answer in the *channel* parameter.

Return Values:

GL_FAILURE, GL_SUCCESS

See Also:

GLEnergyEqn and GLRtnCode

GL_find_regnpks

Definition:

```
GLRtnCode GL_find_regnpks(GLChanRange     region,
                          GLTypedPeaks    *allpeaks,
                          GLTypedPeaks    *pks_in_rgn);
```

Description:

GL_find_regnpks determines which peak in the *allpeaks* list is in the given region. Each of these peaks is returned in the *pks_in_rgn* parameter. The user must provide enough space for the answer in this parameter.

This function is commonly used to prepare the list of peaks passed into the GL_fitregn function.

Return Values:

GL_OVRLMT, GL_SUCCESS

See Also:

GLChanRange, GLTypedPeaks, and GLRtnCode

GL_fitrec_alloc

Definition:

GLFitRecord *GL_fitrec_alloc(int listlength);

Description:

GL_fitrec_alloc allocates memory for the GLFitRecord structure. The user should indicate the maximum number of peaks allowed in a region with the *listlength* parameter. This should match or exceed the value passed into the GL_fitregn function with the GLFitParms->max_npeaks parameter. The value of GLFitRecord->listlength is set properly in the returned structure.

Return Values:

If successful, GL_fitrec_alloc returns the address of the new GLFitRecord structure. If GL_fitrec_alloc fails, it returns NULL.

See Also:

GLFitParms, GLFitRecord, GL_fitrec_free, GL_fitregn

GL_fitrec_free

Definition:

void GL_fitrec_free(GLFitRecord *fitrec);

Description:

GL_fitrec_free frees the memory of *fitrec* that was previously allocated by GL_fitrec_alloc.

See Also:

GL_fitrec_alloc, GLFitRecord

GL_fitregn

Definition:

```
GLRtnCode GL_fitregn(GLChanRange     chanrange,
                     GLSpectrum      spectrum,
                     GLTypedPeaks    pklist,
                     GLFitParms      fitparms,
                     GLEnergyEqn     ex,
                     GLWidthEqn      wx,
                     GLFitRecList    **fitlist);
```

Description:

GL_fitregn fits the region (*chanrange*) of data in the *spectrum* using the peaks indicated in *pklist*, the parameters stored in *fitparms*, and the equations in *ex* and *wx*. It stores the answer in the *fitlist* parameter. The memory allocation for the answer is done inside GL_fitregn. Normally, the user sets *fitlist* equal to NULL before calling GL_fitregn. If *fitlist* is not equal to NULL, GL_fitregn assumes that *fitlist* points to an old list and tries to free it.

Example code:

```
GLFitRecList  *fitlist;
fitlist = NULL;
return_code = GL_fitregn(chanrange, spectrum, pklist, fitparms, ex, wx,
                                                    &fitlist);
```

Return Values:

GL_SUCCESS, GL_BADMALLOC, GL_FAILURE

See Also:

GLChanRange, GLEnergyEqn, GLFitParms, GLFitRecList, GLRtnCode, GLSpectrum, GLTypedPeaks, GLWidthEqn

GL_peaksearch

Definition:

```
GLRtnCode GL_peaksearch(GLChanRange    chanrange,
                        GLWidthEqn     wx,
                        int            threshold,
                        GLSpectrum     spectrum,
                        GLPeaks        *peaks);
```

Description:

GL_peaksearch identifies peaks within the indicated channel range of the spectrum. The width equation *wx* is used to prevent choosing overlapping peaks. The *threshold* parameter controls the pruning out of insignificant peaks. Larger thresholds prune the list more. Recommended threshold values are threshold=20 for more pruning, threshold=10 for average pruning, and threshold=5 for less pruning. The user must provide space in the *peaks* parameter for the answer. Running out of space is indicated in the return code.

Return Values:

GL_BADCHNRNG, GL_BADTHRESH, GL_FAILURE, GL_BADPEAKWD, GL_BADMALLOC, GL_OVRLMT,GL_SUCCESS

See Also:

GLChanRange, GLWidthEqn, GLSpectrum, GLPeaks, and GLRtnCode

GL_postprocess

Definition:

```
GLRtnCode GL_postprocess(GLEfficiency    *efficy,
                         GLSummary       *summary,
                         GLPostInfo      *info);
```

Description:

GL_postprocess uses the *efficy* and *summary* parameters to compute the necessary information to fill the *info* parameter. The user must provide enough space in *info* for the answer. This means that *info-> listlength* should be ≥ *summary->npeaks*.

Return Values:

GL_OVRLMT, GL_SUCCESS

See Also:

GLEfficiency, GLPostInfo, GLRtnCode, GLSummary

GL_prune_rqdpks

Definition:

```
GLRtnCode GL_prune_rqdpks(GLEnergyEqn     ex,
                          GLWidthEqn      wx,
                          GLPeaks         searchpks,
                          GLEnergyPeaks   curr_rqd,
                          GLEnergyPeaks   *new_rqd);
```

Description:

GL_prune_rqdpks copies *curr_rqd* to *new_rqd*, skipping those peaks that are too close to a peak in the *searchpks* list. This is sometimes used in conjunction with the GL_typed_peaks function to prepare a peaklist for input to the GL_fitregn function.

Return Values:

GL_BADMALLOC, GL_FAILURE, GL_SUCCESS, GL_OVRLMT

See Also:

GLEnergyEqn, GLEnergyPeaks, GL_fitregn, GLPeaks, GLRtnCode, GL_typed_peaks, GLWidthEqn

GL_regnsearch

Definition:

```
GLRtnCode GL_regnsearch(GLChanRange     chanrange,
                        GLWidthEqn      wx,
                        double          threshold,
                        int             irw,
                        int             irch,
                        GLSpectrum      spectrum,
                        GLPeaks         peaks,
                        GLRgnSrchMode   mode,
                        int             maxrgnwid,
                        GLRegions       *regions);
```

Description:

GL_regnsearch identifies regions within the specified channel range that a user might want to pass to the GL_fitregn function later. The width equation *wx* is used to compute a working background, to decide how wide a region should be around each peak in the *peaks* list, and to compute a minimum region width. The maximum region width is controlled with the *maxrgnwid* parameter. Whether regions without peaks are included in the answer is controlled with the *mode* parameter.

The *threshold* parameter controls the pruning out of insignificant regions. Larger thresholds prune the list more. Recommended threshold values are threshold=3 for more pruning, threshold=2 for average pruning, and threshold=1 for less pruning.

The *irw* and *irch* parameters control padding the ends of the regions. The regions are padded in pairs of adjacent ends. If the gap between the two region ends being padded is greater than *irw* × peakwidth, then the region ends are padded with this. Otherwise, the region ends are padded with the gap minus *irch* if this pad is greater than zero. Recommended starting values for these parameters is irw=3 and irch=2.

The user must provide space in the *regions* parameter for the answer. Running out of space is indicated in the return code.

Return Values:

GL_BADCHNRNG, GL_BADIRCH, GL_BADIRW, GL_BADTHRESH, GL_BADMALLOC, GL_OVRLMT, GL_SUCCESS

See Also:

GLChanRange, GLPeaks, GLRegions, GLRgnSrchMode, GLRtnCode, GLSpectrum, GLWidthEqn

GL_set_sigcount

Definition:

void GL_set_sigcount(GLSpectrum *spec);

Description:

GL_set_sigcount uses the values in the *spec->count* array to initialize the *spec->sigcount* array. It uses small count correction for counts ≤ 10 based on G. W. Phillips, NIM 153 (1978), p. 449. The user must provide space for the answer in *spec->sigcount*.

See Also:

GLSpectrum

GL_spline

Definition:

void GL_spline(GLEfficiency *efficiency);

Description:

GL_spline uses the abscissas and ordinates found in *efficiency* to calculate coefficients. This is a way for the user to prepare *efficiency->coeff* for use by GL_postprocess. The user must ensure there is space for the answer in *efficiency->coeff*.

See Also:

GL_postprocess, GLEfficiency

GL_summ_alloc

Definition:

GLSummary *GL_summ_alloc(int listlength);

Description:

GL_summ_alloc allocates memory for the GLSummary structure. The user should indicate the maximum number of peaks allowed in a region with the *listlength* parameter.

Return Values:

If successful, GL_summ_alloc returns the address of the new GLSummary structure. If GL_summ_alloc fails, it returns NULL.

See Also:

GLSummary, GL_summ_free

GL_summ_free

Definition:

void GL_summ_free(GLSummary *summary);

Description:

GL_summ_free frees the memory of *summary* that was previously allocated by GL_summ_alloc.

See Also:

GLSummary, GL_summ_alloc

GL_summarize

Definition:

```
GLRtnCode GL_summarize(GLSpectrum    spectrum,
                       GLFitRecord   *fit,
                       GLSummary     *summary);
```

Description:

GL_summarize summarizes the information found in *fit* and stores it in *summary*. The user must provide space in the *summary* parameter for the answer.

Return Values:

GL_BADMALLOC, GL_OVRLMT, GL_SUCCESS

See Also:

GLSpectrum, GLFitRecord, GLSummary, and GLRtnCode

GL_typed_peaks

Definition:

```
GLRtnCode GL_typed_peaks (GLPeaks        *normal,
                          GLEnergyPeaks  *required,
                          GLEnergyEqn    ex,
                          GLTypedPeaks   *combined);
```

Description:

GL_typed_peaks combines a list of *normal* peaks and *required* peaks into a list of typed peaks. Since this list expresses every peak in terms of channel, the energy equation *ex* is used to compute the channel of each *required* peak.
The user must provide space in the *combined* parameter for the answer.

Return Values:

GL_OVRLMT, GL_SUCCESS

See Also:

GLPeaks, GLEnergyPeaks, GLTypedPeaks, and GLRtnCode

GL_wcalib

Definition:

```
GLRtnCode GL_wcalib(int          count,
                    double       *channel,
                    double       *wid,
                    double       *sigw,
                    GLWidEqnMode mode,
                    GLboolean    weighted,
                    GLWidthEqn   *wx);
```

Description:

GL_wcalib uses channel-width-uncertainty triplets to compute a peakwidth equation. The number of triplets is indicated in the *count* parameter. The triplets are passed via the arrays *channel*, *wid*, and *sigw*. The *mode* parameter indicates what kind of equation to produce. See GLWidEqnMode for the current choices. The *weighted* parameter indicates whether to use the uncertainties in the sigw array (weighted = GL_TRUE) or use the constant *1* for the uncertainties (weighted = GL_FALSE).

The user must provide space for the answer in the *wx* parameter.

Return Values:

GL_BADMALLOC, GL_BADCALBLINF, GL_SUCCESS

See Also:

GLboolean, GLWidthEqn, GLRtnCode, GLWidEqnMode

Appendix C

Gauss Algorithms Types

GLCCType

Definition:

```
typedef enum
{
GL_CC_LARGER = 0,
GL_CC_SMALLER = 1,
GL_CC_LARGER_INC = 2
} GLCCType;
```

Description:

GLCCType is an enumeration of the legal values for the *cc_type* member of GLFitParms. *cc_type* is used in conjunction with the *pkwd_mode* member of GLFitParms to choose values for the convergence criteria *ftol* and *xtol*, which are inputs to the Levenberg-Marquardt algorithm used by GL_fitregn. The comments in the algorithm source indicate that *ftol* measures the relative error desired in the sum of squares and *xtol* measures the relative error desired in the approximate solution. A recommended starting value for *cc_type* is GL_CC_LARGER.

	pkwd_mode	
cc_type	GL_PKWD_VARIES	GL_PKWD_FIXED
GL_CC_LARGER	ftol = .001 xtol = .003	ftol = .01 xtol = .01
GL_CC_SMALLER	ftol = .001 xtol = .001	ftol = .001 xtol = .001
GL_CC_LARGER_INC	ftol = .001 xtol = .003	ftol = .001 xtol = .003

See Also:

GLFitParms, GLPkwdMode, GL_fitregn

GLChanRange

Definition:

```
typedef struct
{
int        first;
int        last;
} GLChanRange;
```

Description:

GLChanRange is a structure type used to describe a region in terms of a range of channels that fall in the spectrum defined in a GLSpectrum type. The following relation between a region defined with GLChanRange and a spectrum defined with GLSpectrum is expected:

GLSpectrum.firstchannel ≤ GLChanRange.first ≤ GLChanRange.last ≤ (GLSpectrum.firstchannel + GLSpectrum.nchannels - 1)

See Also:

GL_find_regnpks, GL_fitregn, GL_peaksearch, GL_regnsearch, GLFitRecord, GLCurve, GLRegions, GLSpectrum

GLCurve

Definition:

```
typedef struct
{
int         listlength;       /* should be ≥ npoints */
GLChanRange chanrange;        /* indicates the region */
int         nplots_per_chan;  /* how many plots between channels + 1 */
int         npoints;          /* region width times nplots_per_chan */
int         npeaks;           /* how many component peaks to graph */
double      *x_offset;        /* array of x coordinates used by
                                 all following arrays */
double      **fitpeak;        /* array of component peak curves to be
                                 graphed */
double      *fitcurve;        /* fitcurve to be graphed */
double      *back;            /* background to be graphed */
double      *resid;           /* residuals of the fitcurve */
} GLCurve;
```

Description:

GLCurve is a structure type used to hold coordinates for graphing the fit *fitcurve*, background *back*, and peak component fitcurves *fitpeak[i]* of a fitted region. The structure type also holds residuals. *listlength* indicates how much storage is available for each of the arrays *fitpeak[i]*, *fitcurve*, *back*, and *resid*. *npeaks* indicates how many peak component fitcurves are in the *fitpeak* array. The user must allocate adequate storage in *x_offset*, *fitpeak*, *fitcurve*, *back*, and *resid*, and then set *listlength* appropriately.

See Also:

GL_curve, GL_curve_alloc, GL_curve_free, GLChanRange

GLEfficiency

Definition:

```
#define GL_MAX_EFFICY   100
#define GL_NUM_COEFFS  3

typedef struct
{
int          neff;                          /* number of entries in arrays */
double       en[GL_MAX_EFFICY];             /* abscissas */
double       eff[GL_MAX_EFFICY];            /* ordinates */
double       coeff[GL_MAX_EFFICY][GL_NUM_COEFFS];
                                            /* spline coefficients */
} GLEfficiency;
```

Description:

GLEfficiency is a structure type to hold information used in post processing. *en* holds the abscissas of the knots in strictly increasing order. *eff* holds the ordinates of the knots. *coeff* holds spline coefficients as calculated by GL_spline.

See Also:

GL_postprocess, GL_spline

GLEgyEqnMode

Definition:

```
typedef enum
{
GL_EGY_LINEAR,
GL_EGY_QUADRATIC
} GLEgyEqnMode;
```

Description:

GLEgyEqnMode is an enumeration of the legal values for the *mode* member of GLEnergyEqn.

See Also:

GL_ecalib, GLEnergyEqn

GLEnergyEqn

Definition:

```
typedef struct
{
double          a;          /* coefficient of equation */
double          b;          /* coefficient of equation */
double          c;          /* coefficient of equation */
double          chi_sq;     /* the χ²R of the calibration */
GLEgyEqnMode    mode;       /* the form of the equation */
} GLEnergyEqn;
```

Description:

GLEnergyEqn is a structure type that holds the information needed to describe an energy equation. If the coefficients are a result of calibration, the *chi_sq* member is the χ^2_R of that calibration. Otherwise, the *chi_sq* member is zero. For the equations in the table below, x is the spectral channel, and e(x) is energy as a function of channel.

mode	equation form
GL_EGY_LINEAR	$e(x) = a + bx$
GL_EGY_QUADRATIC	$e(x) = a + bx + cx^2$

See Also:

GL_chan_to_e, GL_e_to_chan, GL_ecalib, GL_fitregn, GL_prune_rqdpks, GL_typed_peaks, GLEgyEqnMode, GLFitRecord

GLEnergyPeaks

Definition:

```
typedef struct
{
int         listlength;     /* storage available in the energy array */
int         npeaks;         /* number of entries stored in the energy array */
double      *energy;        /* array of peaks */
} GLEnergyPeaks;
```

Description:

GLEnergyPeaks is a structure type that holds a list of peaks expressed in terms of energy. The user must allocate adequate storage in *energy* and then set *listlength* appropriately.

See Also:

GL_prune_rqdpks, GL_typed_peaks

GLFitInfo

Definition:

```
typedef struct
{
int             listlength;     /* storage length of peakinfo */
double          intercept;      /* background intercept at last channel */
double          slope;          /* background slope */
double          step_height;    /* unused - was fraction of gaussian height */
double          avg_width;      /* full-width-half-max in channels */
int             npeaks;         /* number of GLPeakInfos stored in peakinfo */
GLPeakInfo      *peakinfo;      /* fit information of each peak */
} GLFitInfo;
```

Description:

GLFitInfo is a structure type that holds information about peak location, width, height, and background that is used to initiate a fit cycle from within the GL_fitregn function. It is overwritten with the answer. The background can be calculated from the *intercept* and *slope*. The peakwidth is calculated from *avg_width* and from information in *peakinfo*. The peak location and height are in *peakinfo*. This structure type can also be used to record the uncertainty of each member of a corresponding GLFitInfo.

See Also:

GL_fitregn, GLFitRecord

GLFitParms

Definition:

```
typedef struct
{
int             ncycle;         /* max # of fit cycling allowed */
int             nout;           /* max # of best cycles to return */
int             max_npeaks;     /* max # of peaks allowed in a fit */
GLPkwdMode      pkwd_mode;      /* controls whether peakwidth can vary */
GLSplitMode     split_mode;     /* controls how items varied in fit cycle */
GLCCType        cc_type;        /* controls input to Levenberg-Marquardt
                                   algorithm */
float           max_resid;      /* max residual to determine whether to
                                   recycle */
} GLFitParms;
```

Description:

GLFitParms is a structure type that holds the parameters passed in to GL_fitregn. These parameters control the fitting, the recycling, and the reporting. If the maximum residual of the fit is greater than *max_resid*, then GL_fitregn will try to recycle (add a peak and fit again). But if adding a peak would cause the peak count to exceed *max_npeaks*, then it won't recycle. If recycling would mean that the number of cycles exceeds *ncycle*, then it won't recycle. Or, if the location of the maximum residual is too close to an existing peak, then it won't recycle. *nout* indicates the maximum number of cycles the user wants returned by GL_fitregn. The *nout* best ones are returned.

Good initial values are:

ncycle=5, nout=3, max_npeaks=10, pkwd_mode=GL_PKWD_VARIES, split_mode = GL_SPLIT_NONE, cc_type = GL_CC_LARGER, max_resid = 2.

See Also:

GL_fitregn, GLCCType, GLPkwdMode, GLFitRecord, GLSplitMode

GLFitRecList

Definition:

```
typedef struct fitreclist
{
GLFitRecord     *record;    /* structure holding one fit cycle answer */
struct fitreclist   *next;  /* pointer to next record in linked list */
} GLFitRecList;
```

Description:

GLFitRecList is a linked list structure type that holds a list of GLFitRecords returned by GL_fitregn.

See Also:

GL_fitregn, GLFitRecord

GLFitRecord

Definition:

```
#define GL_COVARR_DIM  3

typedef struct
{
int             cycle_number;     /*which cycle created this record*/
GLChanRange     used_chanrange;   /*what region was passed in */
GLFitParms      used_parms;       /*what parameters were passed in*/
GLEnergyEqn     used_ex;          /*what energy equation was passed in */
GLWidthEqn      used_wx;          /*what width equation was passed in */
int             lmder_info;       /*answer from Levenberg-Marquardt alg*/
double          chi_sq;           /* χ²R of this fit */
GLFitInfo       fitinfo;          /*description of fit */
GLFitInfo       uncertainty;      /*corresponding fit uncertainties */
double          (*covarr)[GL_COVARR_DIM];
                                  /* computed from jacobian */
} GLFitRecord;
```

Description:

GLFitRecord is a structure type that holds information about a single fit cycle of the specified region. The “used” members record what information was passed into the GL_fitregn function. *covarr* is a two-dimensional array with the first dimension = fitinfo->listlength. Within the GL_fitregn function, this information is used as a starting point for fitting, and later overwritten with the answer. This structure type can also be used to record the *uncertainty* of each member of a corresponding GLFitRecord.

See Also:

GL_fitrec_alloc, GL_fitrec_free, GL_fitregn, GLChanRange, GLEnergyEqn, GLFitInfo, GLFitParms, GLFitRecList, GLWidthEqn

GLPeakInfo

Definition:

```
typedef struct
{
double      height;         /* the height of the peak */
double      centroid;       /* the location of the peak relative to the region */
double      addwidth_511;   /* additional peakwidth due to the 511keV peak */
GLPeakType  peak_type;      /* whether the peak location is fixed or not */
} GLPeakInfo;
```

Description:

GLPeakInfo is a structure type that holds information about a single peak in a single fit cycle of a region. Within the GL_fitregn function, this information is used as a starting point for fitting, and is later overwritten with the answer. This structure type can also be used to record the uncertainty of each member of a corresponding GLPeakInfo.

If the fit algorithm returns zero for the *height* member, this indicates that the peak was deleted during one of the cycles of the fit.

The value of the *centroid* member ranges from 1 to "region width," where 1 indicates the first channel in the region.

See Also:

GL_fitregn, GLFitInfo, GLPeakType

GLPeaks

Definition:

```
typedef struct
{
int         listlength;     /* amount of space in channel */
int         npeaks;         /* number of peaks stored in channel */
double      *channel;       /* array of peaks */
} GLPeaks;
```

Description:

GLPeaks is a structure type that holds a list of peaks expressed in terms of channel locations. The user must allocate adequate storage in *channel* and then set *listlength* appropriately.

See Also:

GL_peaksearch, GL_prune_rqdpks, GL_regnsearch, GL_typed_peaks, GLTypedPeaks

GLPeakType

Definition:

```
typedef enum
{
GL_PEAK_NORMAL = 0,
GL_PEAK_RQD = 1
} GLPeakType;
```

Description:

GLPeakType is an enumeration of the legal values for members of both GLTypedPeaks and GLPeakInfo. It indicates whether the peak location is fixed or not.

GLPeakType	result
GL_PEAK_NORMAL	channel location not fixed
GL_PEAK_RQD	channel location fixed

See Also:

GLTypedPeaks, GLPeakInfo

GLPkwdMode

Definition:

```
typedef enum
{
GL_PKWD_VARIES = 0,
GL_PKWD_FIXED = 2
} GLPkwdMode;
```

Description:

GLPkwdMode is an enumeration of the legal values for the *pkwd_mode* member of GLFitParms. It controls whether peak width can vary during a fit cycle, where peak width is computed from GLFitInfo->avg_width and GLPeakInfo[j]->addwidth_511. GLPkwdMode also works with GLCCType to control input to the Levenberg-Marquardt algorithm.

Table showing when GLFitInfo->avg_width is allowed to vary:

	count of regular peaks	
	none	at least 1
GL_PKWD_VARIES	not allowed	allowed
GL_PKWD_FIXED	not allowed	not allowed

Table showing when GLPeakInfo[j]->addwidth_511 is allowed to vary: (GLPeakInfo[j]->addwidth_511 must be nonzero to be allowed to vary, which occurs when the jth peak is tagged as a 511keV peak.)

	count of any peaks	
	less than 2	at least 2
GL_PKWD_VARIES	not allowed	allowed
GL_PKWD_FIXED	allowed	allowed

Note that the software prevents allowing both peakwidth parameters, GLFitInfo->avg_width and GLPeakInfo[j]->addwidth511, to vary when there is only one peak. As peaks are added or deleted for recycling, the tables above are followed to update whether the *avg_width* and *addwidth_511* members are allowed to vary.

See Also:

GLCCType, GLFitParms

GLPostInfo

Definition:

```
typedef struct
{
int         listlength;     /* storage length of each arrays */
int         npeaks;         /* number of items stored in each array */
double      *efficiency;    /* detector efficiency at each peak */
double      *intensity;     /* ratio of peak area to efficiency */
double      *sigi;          /* intensity uncertainty */
} GLPostInfo;
```

Description:

GLPostInfo is a structure type to hold the answer from GL_postprocess.

See Also:

GL_postprocess

GLRegions

Definition:

```
typedef struct
{
int         listlength;     /* amount of storage in chanrange */
int         nregions;       /* number of regions stored in chanrange */
GLChanRange *chanrange;     /* array of regions */
} GLRegions;
```

Description:

GLRegions is a structure type to hold a list of regions. The user must set up the space for the *chanrange* array and set *listlength* appropriately.

See Also:

GL_regnsearch, GLChanRange

GLRgnSrchMode

Definition:

```
typedef enum
{
GL_RGNSRCH_ALL,
GL_RGNSRCH_FORPKS
} GLRgnSrchMode;
```

Description:

GLRgnSrchMode is an enumeration of the legal values of the *mode* parameter of GL_regnsearch. This parameter indicates whether to search for any region, or restrict the search to regions that contain one of the peaks also passed into the GL_regnsearch function.

See Also:

GL_regnsearch

GLRtnCode

Definition:

```
typedef enum
{
GL_SUCCESS,
GL_FAILURE,
GL_BADMALLOC,
GL_BADCALBLINF,
GL_BADPEAKWD,
GL_BADCHNRNG,
GL_BADIRCH,
GL_BADIRW,
GL_BADTHRESH,
GL_OVRLMT
} GLRtnCode;
```

Description:

GLRtnCode is an enumeration of the legal values returned by the suite of Gauss Algorithms (except for the functions that return pointers to structure types).

GL_SUCCESS - no problems in execution of procedure
GL_FAILURE - unspecified problem in execution of procedure
GL_BADMALLOC - failure to allocate temporary workspace in memory; procedure returned without completing.
GL_BADCALBLINF - in calibration, error returned by linf_()
GL_BADPEAKWD - in any procedure, GL_chan_to_w() returned error and recovery was not possible.
GL_BADCHNRNG - the channel range is illegal or inconsistent with spectrum
GL_BADIRCH - the value of irch passed in is ≤ 0
GL_BADIRW - the value of irw passed in is ≤ 0
GL_BADTHRESH - the value of threshold passed in is < 0
GL_OVRLMT - more peaks or regions were found than structure could hold; the extra peaks or regions are ignored; the returned list contains the items found up to the limit.

See Also:

GL_chan_to_w, GL_curve, GL_ecalib, GL_e_to_chan, GL_find_regnpks, GL_fitregn, GL_peaksearch, GL_postprocess, GL_prune_rqdpks, GL_regnsearch, GL_summarize, GL_typed_peaks, GL_wcalib

GLSpectrum

Definition:

```
typedef struct
{
int                listlength;      /* amount of storage in count & sigcount*/
int                nchannels;       /* # of items stored in count & sigcount */
int                firstchannel;    /* a location on the X axis */
unsigned long int  *count;          /* y coordinates of spectrum */
double             *sigcount;       /* uncertainties of y coordinates */
} GLSpectrum;
```

Description:

GLSpectrum is a structure type to hold the counts per channel of a spectrum. *firstchannel* can be any positive or negative integer. *nchannels* must be positive or zero.

For example, the location of the last channel on the X axis can be calculated from

last_channel = GLSpectrum->firstchannel + GLSpectrum->nchannels - 1

Both the *count* and *sigcount* arrays are indexed from 0 to *nchannels* - 1.

As another example, to retrieve the count at channel “J,” compute

index = J - GLSpectrum->firstchannel

to access GLSpectrum->count[index].

See Also:

GLCurve, GL_fitregn, GL_peaksearch, GL_regnsearch, GL_set_sigcount, GL_summarize

GLSplitMode

Definition:

```
typedef enum
{
GL_SPLIT_NONE = 0,
GL_SPLIT_ALLOWED = 1
} GLSplitMode;
```

Description:

GLSplitMode is an enumeration of the legal values for the *split_mode* member of GLFitParms.

If split_mode = GL_SPLIT_NONE, then GL_fitregn calls the Levenberg-Marquardt program only once, passing in a list of all the parameters that are allowed to vary.

If split_mode = GL_SPLIT_ALLOWED, then GL_fitregn calls the Levenberg-Marquardt program five times. The first and third time, only the linear parameters (intercept, slope, peak heights, avg_width) are passed in to be varied. The second and fourth time, only the nonlinear parameters (peak centroids and addwidth_511s) are passed in to be varied. The last time, all parameters allowed to vary are passed in. The intent of the GL_SPLIT_ALLOWED mode is to allow the fast linear least squares fitting method to improve these parameter estimates a great deal before the nonlinear fitting is done. But this mode takes more time and so is only recommended to investigate the fitting process itself.

See Also:

GLFitParms

GLSummary

Definition:

```
typedef struct
{
int         listlength;     /* amount of storage available for each array */
int         npeaks;         /* number of items stored in each array */
double      ratio;          /* ratio of summation area to integral area */
double      *channel;       /* array of peak centroids */
double      *sigc;          /* array of centroid uncertainties */
double      *height;        /* array of peak heights */
double      *sigh;          /* array of height uncertainties */
double      *wid;           /* array of peak widths */
double      *sigw;          /* array of width uncertainties */
double      *area;          /* array of peak areas */
double      *siga;          /* array of area uncertainties */
double      *energy;        /* array of peak energies */
double      *sige;          /* array of energy uncertainties */
} GLSummary;
```

Description:

GLSummary is a structure type for holding the answer from GL_summarize.

See Also:

GL_postprocess, GL_summ_alloc, GL_summ_free, GL_summarize

GLTypedPeaks

Definition:

```
typedef struct
{
GLPeaks         peak;          /* list of peaks */
GLPeakType      *type;         /* array of type of each peak in list above */
} GLTypedPeaks;
```

Description:

GLTypedPeaks is a structure type for holding a list of peaks along with the type of each peak. In addition to providing the necessary storage in the *peak* member, the user is expected to provide enough storage in the *type* member for up to *peak->listlength* entries.

See Also:

GL_find_regnpks, GL_fitregn, GL_typed_peaks, GLPeaks, GLPeakType

GLWidEqnMode

Definition:

```
typedef enum
{
GL_WID_LINEAR,
GL_WID_SQRT
} GLWidEqnMode;
```

Description:

GLWidEqnMode is an enumeration of the legal values for the *mode* member of GLWidthEqn.

See Also:

GL_wcalib, GLWidthEqn

GLWidthEqn

Definition:

```
typedef struct
{
double          alpha;      /* coefficient of equation */
double          beta;       /* coefficient of equation */
double          chi_sq;     /* the χ²R of the calibration */
GLWidEqnMode    mode;       /* the form of the equation */
} GLWidthEqn;
```

Description:

GLWidthEqn is a structure type that holds the information needed to describe a peak width equation. If the coefficients are a result of calibration, the the *chi_sq* member is the χ^2_R of that calibration. Otherwise, the *chi_sq* member is zero. For the equations in the table below, *x* is the spectral channel, and *w(x)* is peak width as a function of channel.

mode	equation form
GL_WID_LINEAR	$w(x) = \alpha + \beta x$
GL_WID_SQRT	$w(x) = \sqrt{\alpha + \beta x}$

See Also:

GL_chan_to_w, GL_fitregn, GL_peaksearch, GL_prune_rqdpks, GL_regnsearch, GL_wcalib, GLFitRecord, GLWidEqnMode

GLboolean

Definition:

```
typedef enum
{
GL_FALSE = False,
GL_TRUE = True
} GLboolean;
```

Description:

GLboolean is an enumeration of *true* and *false* for use with the Gauss Algorithms.

See Also:

GL_ecalib, GL_wcalib