ELSEVIER

Computer Physics Communications 112 (1998) 183–190

Computer Physics Communications

JavaFit: a platform independent program for interactive nonlinear least-squares fitting using the Levenberg–Marquardt method

A.W. Robinson [1]

Nanoscale Physics Research Laboratory, School of Physics and Astronomy, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK

Received 1 November 1997; revised 2 April 1998

Abstract

The JavaFit program is a package for carrying out interactive nonlinear least-squares fitting to determine the parameters of physical models from experimental data. It has been conceived as a platform independent package aimed at the relatively modest computational needs of spectroscopists, where it is often necessary to determine physical parameters from a variety of spectral lineshape models. The program is platform independent, provided that a Java runtime module is available for the host platform. The program is also designed to read a wide variety of data in ASCII column formats produced on DOS, Macintosh and UNIX platforms. © 1998 Elsevier Science B.V.

PACS: 0.2.60.Ed; 82.80.Pv; 82.80.Ch
Keywords: Interactive; Nonlinear least-squares; Levenberg–Marquardt; Optimisation; Spectroscopy; Model fitting

PROGRAM SUMMARY

Title of program: JavaFit

Catalogue identifier: ADIL

Program Summary URL:
http://www.cpc.cs.qub.ac.uk/cpc/summaries/ADIL

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland

Licensing provisions: none

Computers: designed for any machine capable of running Java, developed on Macintosh 7200/75, also run on PC-Pentium-100

Operating systems under which the program has been tested: MacOS 7.5.5, Linux 2.0.27 (Red Hat release 4.6)

Programming language used: Java

Memory required to execute with typical data: case dependent

No. of bytes in distributed program, including test data, etc.: 92626

Distribution format: uuencoded compressed tar file

Other software required: A Java runtime interpreter, or the Java Development Kit, version 1.0.2

Keywords: Interactive, nonlinear least-squares, Levenberg–Marquardt, optimisation, spectroscopy, model fitting

Nature of physical problem
Interactive least-squares fitting of analytical models of spectra lineshape functions to electron spectroscopy spectra.

Method of solution
The user must supply a subroutine which calculates theoretical

[1] E-mail: andrew.robinson@physics.org

PII S0010-4655(98)00052-6

values of the quantities to be fitted and their first partial derivatives to the adjustable parameters.

Typical running time
Problem dependent.

LONG WRITE-UP

1. Introduction

One of the most frequent tasks which the spectroscopist is faced with, is the fitting of experimental data to trial lineshapes. There may be various reasons for this, such as the necessity to determine the area under the lineshape, to determine the width of spectral functions, in order to understand the physical processes which lead to line broadening, or to deconvolute overlapping lineshapes, when the instrumental resolution is insufficient to distinguish between closely spaced peaks. In general this type of procedure typically has a data set ranging from 50–1000 points, equally spaced in the x-dimension (usually an energy or frequency, which is set by the instrumentation), and with an intensity measurement as the y-data. This data has been frequently fitted using the Levenberg–Marquardt method [1,2]. One of the main difficulties with this analysis has often been the lack of a platform independent package to do this type of fitting on. As most experimental systems are run by a variety of microcomputers, including Macintosh and PC-compatible types, it has often been difficult to transfer fitting programs between laboratories. A further complication is the file format used by various experimental groups, research laboratories and experimental facilities, which, while generally being of columns of data in ASCII format, often have a bewildering variation in terms of number of header or footer lines, and in data separation characters.

The JavaFit program is designed to provide the spectroscopist with a data-fitting tool which may be used on many different machines, but which will always run in the same manner, even though the "look and feel" of the program is dependent on the operating system. It is designed to read a wide variety of ASCII columnar format data files, produced by UNIX, PC and Macintosh computers without any further processing of the data to remove or change line termination characters. This can be a very time consuming process for the working scientist, especially when a large number of files have to be processed.

The user has to provide a function that simulates the particular spectral lineshape being modelled. In the particular case of the JavaFit implementation described here, the lineshape is that of multiple spin-orbit split doublets with a Voigt line profile [3]. We have used this lineshape to model photoemission spectra from the Silicon 2p photoemission level for a Silicon layer adsorbed on a Copper (111) surface [4] and test data is provided from this adsorption system.

The approach to solution of the problem is an interactive graphical one, as the user can use their intuition and knowledge to determine trial solutions which have physically reasonable values. The data is read in and displayed on screen. The user may perform various normalisation operations. JavaFit allows for normalisation to a third data column, which is common in synchrotron photoelectron spectroscopy, where the intensity of the stimulating radiation may fall over the length of time taken to carry out the measurement. In addition there may be a requirement to eliminate the inelastic scattering of electrons in the tail of the photoelectron spectrum, often known as removal of a Shirley type background [5]. The user then enters trial parameters into a dialogue box, and compares the theoretical lineshape generated with the actual data. Once a reasonable agreement between experiment and a parameter set is reached, then the Levenberg–Marquardt fitting is invoked, producing a final set of parameters, which represents a model of the experimental results.

2. Method of solution

The method of solution of a nonlinear least-squares fit is by the Levenberg–Marquardt method [1,2]. The calculation engine is a Java implementation of an established C/FORTRAN fitting technique [6]. Using an object-oriented approach, the whole of the fitting procedure is encapsulated in a Java class, *LevMarq*, with methods available for entry of data and access to the fitting parameters and the resultant fit to the model. The user needs to define the fitting function according to the problem on hand. An instance of the fitting class *LevMarq* can then be declared within the main program.

In the example code given, the data is fitted to a

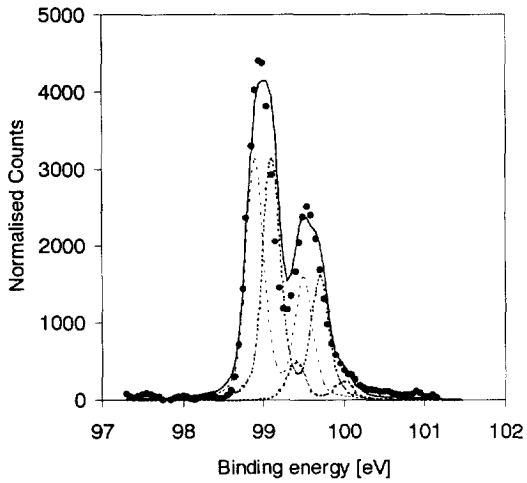

Fig. 1. Experimental data for the Silicon 2p photoemission feature (dots) and the trial function before fitting (solid lines) using the parameters in Table 1, column 2. The three component peaks are plotted as dotted lines.

lineshape that consists of two Voigt profiles. This simulates the photoelectron spectrum obtained from core level electron orbitals, which are split into 2 components by spin-orbit interactions. In the model, the lineshape is determined by the lifetime of the core-hole produced by the ejection of the photoelectron, which has a Lorentzian shape, and a Gaussian component which is due to instrumental broadening from the energy resolution of the stimulating radiation and the energy resolution of the kinetic energy analyser employed. For core levels, the lifetime broadening can be assumed to be equal for both components, as can the instrumental resolution. Generally, there is a fixed intensity ratio (the Branching Ratio) between the two components. Thus the parameters which need to be fitted are the Lorentzian Full Width at Half Maximum (FWHM), the Gaussian FWHM, the intensity of the primary peak, the branching ratio, the energy position (on the x-axis) of the primary peak and the energy (x-axis) offset between the primary and secondary peak. The parameters are shown schematically in Fig. 1.

In the Levenberg–Marquardt method, it is also necessary to quantify the variance in the y-values of the experimental data. Generally, it is not possible to measure the experimental uncertainty at every point in a spectrum, so some degree of approximation has to be carried out. In the present case, the variance is assumed to be equal at every point. As the experimental data in this instance consists of peaks surrounded by a flat background signal, the variance is calculated by computing the standard deviation of the last 10% of points in the spectrum, where the background is flat, and the points should vary around a mean value.

3. Program structure

3.1. Components

The program components that are required to run the JavaFit package are

JavaFit.class
LevMarq.class
ImportDialog.class
GetDoubles.class
MessageBox.class
ExTextField.class
SOSParameters.class

These are the compiled versions of the source code files

JavaFit.java
LevMarq.java
ImportDialog.java
GetDoubles.java
MessageBox.java
ExTextField.java
SOSParameters.java

Note that because of the platform independent nature of Java, the class files as supplied should be able to run on any platform for which there is a Java runtime module (version 1.0.2) installed. In practice, the user will almost certainly want to modify some of the source code in order to modify the fitting function for the physical situation that is being modelled. Running JavaFit is a matter of invoking the Java runtime module on the JavaFit class.

3.2. JavaFit class

The main component of the JavaFit program is the *JavaFit class*. This class is derived from the *Frame* class and is the general driver routine, which contains data arrays, the *Paint* routine for screen display, and class declarations and instance creation for the *Lev-*

Marq fitting class. Data holding arrays are provided in this class because it is quite typical to perform normalisation and background subtraction on data before fitting. As the Levenberg–Marquardt fitting class was devised as a general purpose and self-contained fitting engine, it does not include methods to carry out these specialised normalisations. This makes the *LevMarq* class more compact and portable, but has the drawback that prefitting normalisation requires the data to be stored in the main program, pre-normalised, and then loaded into the *LevMarq* data storage arrays so that fitting can be carried out. The size of the data storage arrays is set by the ARRAYSIZE integer variable, and is set to 200 in the current JavaFit implementation.

The display of *JavaFit* comprised of two components, the main frame, which displays the loaded data and any modelled fit to that data, and a dialogue box, which contains the values of the parameters used in the fitting model. The user enters values into the dialogue box, and presses the update button to replot the screen display with the new model. The dialogue box is created from a separate class.

The *Paint* method is used to display both the entered data and the fit derived by applying the current model parameter values to the model function. In the present model, we allow for the possibility of several overlapping lineshapes, and the paint routine will plot up the total fit together with the individual components. This is a complication that the user will not necessarily need, and appropriate comments are made in the source code regarding which portions of Paint are customised for the multi-component display. There are also several methods called from *Paint* that plot the data and graph axes.

The *ReadFile* method is designed to parse complex ASCII data in columns, including floating point values using the exponential format. It is also designed to deal with the different line termination characters used by MS-DOS, UNIX and MacOS operating systems. It also allows for there to be ASCII text comments in headers before the numerical data, and footers, after the numerical data.

The *handleEvent* method deals with all mouse input, window and menu commands. Most of the program functionality is called from this method.

The *SingleFuncs* method is a variant of the *funcs* method in the *LevMarq* class. It is present because in the fitting model supplied, it is possible to have up to three doublet Voigt profiles present. It is highly desirable to be able to plot out the individual components on the screen in order to see the relative sizes, and also to be able to numerically integrate the area under each component to find the relative areas. The *SingleFuncs* method is derived from the *funcs* method, but is considerably simplified, as it requires only one component to be calculated, and does not require the calculation of the partial derivatives of the fitted function.

The *CalculateSigma* method is used to determine the value of the variance for each experimental data point. The approximation used is to assume that the last 10% of data points in the experimental spectrum lie on a flat background. The variance can be calculated from these points. The same value for the variance is then assigned to each experimental data point using the *SetSigma* method in the *LevMarq* class.

There are a number of graphical methods for plotting the data on screen, and plotting x and y-axes. These are

```
void DrawXAxis(Graphics g)
void DrawXAxis(Graphics g, double interval)
void DrawYAxis(Graphics g)
void DrawYAxis(Graphics g, double interval)
int GetYCoordinate(double dValue)
int GetXCoordinate(double dValue)
void SetScreenSize(int x, int y)
void SetPlottingLimits(double x_min,
double x_max, double y_min, double y_max)
double FindTicks(double AxisMin, double AxisMax)
```

Once the data is loaded into the data arrays defined in JavaFit, it may be necessary to manipulate it to normalise for the intensity of the stimulating radiation, and to remove the effects of inelastic scattering from the background. These functions are implemented in the methods

```
void Normalise()
void SubtractShirleyBackground()
```

After data has been normalised it may then be loaded into the Levenberg–Marquardt fitting class *LevMarq*, to carry out the fitting. After the fitting has been carried out, the data may be saved into two ASCII files. The first file is a four-column file containing the experimental x, y and sigma data, and the calculated fit in the fourth column, respectively. The second file, with the identifier *.par* appended to the

name of the first file, contains text stating the values of each of the parameters used in the calculated fit, and the integrated areas under each component in a multi-component fit. The methods used are

void WriteFile (String filename)
void WriteParameterFile (String filename)

3.3. LevMarq class

LevMarq is a generalised class that totally encapsulates the Levenberg–Marquardt fitting procedure. It includes data storage arrays and methods for storage and retrieval of experimental and calculated data, and the parameters used in the model function.

The user has to define the functional form of the fitting profile in the method *funcs*, which is generally an equation of form $y = f(a_1 \ldots a_n, x)$, where a_1 to a_n are the parameters which are to be varied in the fitting procedure. In order to keep compatibility between functions developed for programs using the C or FORTRAN routines from which it is derived [6], the *funcs* routine uses the same variable name conventions. The values of the fitting parameters 1 to n are held in an array a and the parameter which determines whether it is allowed to float during the fit are held in array *ia*. The routine must also calculate the partial derivative of the function with respect to the parameters. The complications of the original C and FORTRAN routines, which need many arguments to initialise arrays etc., are avoided in the Java implementation, as all of these data storage structures are encapsulated within the *LevMarq* structure. The default implementation of *funcs* is a function that can model multiple doublet Voigt profiles [3].

To change the maximum number of datapoints that can be stored, the integer value MAXPOINTS may be set to an appropriate value. In the current implementation this set to 200. Similarly, the maximum number of adjustable parameters which may be modelled is set using the integer variable MAXPARAMS. This is set to 20 in the program as distributed.

The general procedure for carrying out the fitting is firstly to load the data set into the instance of *LevMarq* using the data input methods:

void SetX(int n, double *xin*)
sets the x data point $x[n]$ to the value *xin*

void SetY(int n, double *yin*)
sets the y data point $y[n]$ to the value *yin*

void SetSigma(int n, double sigin)
sets the variance sigma$[n]$ to sigin

void SetNumData(int n)
sets the number of data points

Loading the values of the parameters into the fitting function, and determining whether these values are allowed to vary (floating) or are fixed during the fitting process, is done with the method

void SetParam(int n, double f, int nf)
puts value f into parameter n and sets the floating/fixed attribute to nf (0 = fixed, 1 = floating).

The initial stage of the fitting procedure is normally done by an iterative process by guessing a trial solution and adjusting the parameters to physically reasonable values that give a reasonable level of agreement with the experimental data. Hence the JavaFit program displays the data that is being fitted, and the fit that corresponds to the parameters entered by the SetParam methods. The user can then adjust the parameters until a reasonable correspondence between the experimental data and the trial set of parameters is obtained. The JavaFit display method *Paint* calls the Levmarq method *FitWithParams*. This method takes the current values of the parameters and applies them to the model function *funcs*.

Once the user has determined that there is a satisfactory agreement between the entered parameters and the data, the fitting method is then called:

double FitData( )

This method then performs the Levenberg–Marquardt fitting, starting with the entered parameters. At each fitting step a value of χ^2 is calculated between the experimental and calculated values. The fitting process continues until the χ^2 values of two successive iterations become very similar. The default convergence criterion is that if that the ratio of the newly calculated chisq value to the previous chisq value is greater than 0.999, then the fitting process is halted. There are methods to alter the convergence criterion for halting the fitting:

void SetConverge(double f)
sets the convergence criterion for fitting

double GetConverge( )
returns the current convergence criterion

Once the fit is completed, then the data may be retrieved using the data access methods:

double GetX(int n)
returns the value of the x data at point n

double GetY(int n)
returns the value of the y data at point n

double GetFit(int n)
returns the value of the fitted data at point n

double GetSigma(int n)
returns the value of the variance at point n

int GetNumData()
returns the number of data points

double GetParam(int n)
returns the value of the nth parameter

double GetChisq()
returns the Chi Squared value from the current fit

double GetCovariance(int n)
returns $(n \times n)$th element in the covariance matrix

double GetVariance(int n)
returns the square root of $(n \times n)$th element in the covariance matrix element

The last two methods are for returning the values in the covariance matrix which is used in the L–M fitting method. This matrix is of size $(n \times n)$, where n is the number of floating parameters in the fitted function. Provided that the estimate of the variance in the experimental y-data is physically reasonable, these covariance values may be used to calculate standard errors on the calculated parameters [3].

3.4. *ImportDialog class*

Contains the code that creates the import dialogue box. This allows the user to define the type of input file. The program expects a multicolumn ASCII format file. There may be comment lines before (the header) or after (the footer) the actual data, depending on the idiosyncrasies of the data file format. In addition to specifying the x and y columns in which the data is to be found, there is the possibility of reading a third column for normalisation data. If no normalisation data is present, the value 0 should be entered in the appropriate field in the dialogue box.

3.5. *GetDoubles class*

This is a routine for reading and parsing lines of text into double precision floating values. It is a modification of a public domain software routine [2].

3.6. *MessageBox class*

Creates a small box, which gives information to the user when time consuming tasks, such file i/o or fitting are taking place.

3.7. *ExTextField class*

A modification of the *TextField* Class (contained in the standard Java AWT library), which allows for easier conversion between strings in text fields contained in the dialogues and double precision floating point values.

3.8. *SOSParameters class*

This class encapsulates a dialogue box for entering the parameters for the fitting routine. The captions and numbers of fields in this class will depend on the type of fitting function used, and the user will need to modify the number of parameters depending on the form of function *funcs*.

3.9. *Platform dependent features*

Despite the intention of making the *JavaFit* program completely platform independent, there is one change which needs to be made to the Macintosh version of this program. This concerns the way in which files are read from the local filing system. The Macintosh implementation requires the addition of a '/' character between the pathname and filename of the input file in order to read from the local disk. This is a Java Development Kit (JDK) bug specifically in the Macintosh implementation of JDK 1.0.2. The correct code according to the JDK documentation is

[2] The original routine was written by T. Farnum (tfarnum@rpa.net). The version supplied has been modified by the author to cope more flexibly with exponential formats in ASCII data.

```
infile = opendlg.getDirectory()
        + opendlg.getFile();
```

This must be modified in the Macintosh version using JDK 1.0.2. to

```
infile = opendlg.getDirectory()
        + ''/'' + opendlg.getFile();
```

There are no other platform dependent features, but it should be noted that the performance of JavaFit on the different platforms may vary considerably. This could be due to the processor speed, the relative efficiency of the operating system and the efficiency of the implementation of the java runtime executable. In the systems tested here, the Macintosh implementation (on a Mac 7200/75 running MacOS 7.5.5) performs the L–M fitting extremely rapidly, but is rather slow on file i/o operations. The Linux version (Pentium P133, running Linux 2.0.27) has fast file i/o, but is rather slower in carrying out the actual fitting procedure. As the difference in fitting performance is an increase in the fitting time from around 1 second on the Macintosh to 3 seconds on the Linux machine, this is not a significant problem. The performance of Java, as an interpreted language, will be inferior to compiled languages such as C or FORTRAN. However, the actual fitting procedures take only a matter of seconds to carry out. The time consuming tasks are actually the interactive setting of the initial modelling parameters before the fitting procedure starts, which is largely determined by the user. Hence JavaFit has sufficient performance to carry out the fitting tasks.

4. Test run

The user should select the "Import" option in the "File" pull-down menu, and check that the characteristics of the data file in the dialogue box are 0 header lines, 0 footer lines, x data from column 1, y data from column 2 and normalisation data from column 0. The dialogue box may then be closed. Then selecting Open from the File pull down menu, the file *jtest.dat* may be read in. The data has already been prenormalised and a background subtracted, so no operations are necessary from the "Background" menu. The fitting parameters dialogue box is then made visible using the "Show" option in the "Parameters" pull-down menu. The trial fitting parameters are then entered into the dialogue and should be set to those given in column 2 of Table 1. The "floating" checkboxes of all parameters should be checked in the dialogue box, except for the branching ratio and the spin-orbit splitting values, which should remain constant at 2.0 and 0.6, respectively, throughout the fitting procedure. The validity of these values is discussed in Ref. [4]. The number of peaks in the dialogue box should be set to 3, as in

Table 1
The values of the 6 parameters needed to model the spin-orbit double Voigt photoemission function

Parameter	Value in trial solution before fitting	Value after L–M fit
Branching ratio	2.0	2.0
Spin-orbit splitting [eV]	0.6	0.6
Lorentzian FWHM [eV]	0.1	0.123
Gaussian FWHM [eV]	0.2	0.186
Intensity of Peak 1	3200	2845.38
Position of Peak 1 [eV]	99.5	99.51
Intensity of Peak 2	3200	1903.6
Position of Peak 2 [eV]	99.7	99.66
Intensity of Peak 3	500	332.94
Position of Peak 3 [eV]	100.0	99.98

The second column contains those used to model the trial fit shown in Fig. 1, the third column contains those obtained after the L–M fitting procedure, shown in Fig. 2.

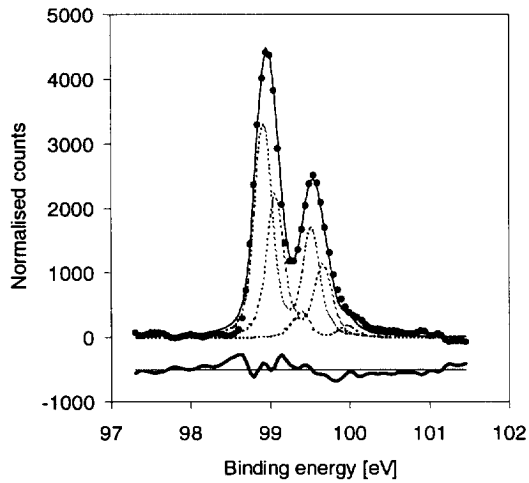

Fig. 2. Experimental data (dots) compared to the fit obtained after the Levenberg–Marquardt fit (solid line). The values of the fitted parameters are given in Table 1, column 3. The three component peaks are plotted as dotted lines. The solid line is the residual, offset by −500 units in y for clarity.

this case we are fitting the data to three doublet peak profiles. Once the "Update" button in the dialogue box is pressed, the resultant screen display, comparing experiment data to the trial fit, is shown in Fig. 1.

The fitting routine may then be invoked by clicking the "Do Fit" menu item in the "Fit" menu. Once this is completed, *JavaFit* returns the calculated parameters to the dialogue box and displays the resultant fit, together with the experimental data, on screen. The calculated values from fitting the test data are given in column 3 of Table 1, and used to plot the final model against the experimental data in Fig. 2. As can be seen, the final calculated fit shows a good level of agreement with the experimental data, despite starting from the relatively poor first guess of Fig. 1.

Acknowledgements

The author is grateful to J. Wilkes for installing and extensively testing the Linux version of the program and providing many helpful comments. The author also thanks C.E.J. Mitchell for a critical read-through of the manuscript.

References

[1] K. Levenberg, Quart. Appl. Math. 2 (1944) 164.
[2] D. Marquardt, SIAM J. Appl. Math 11 (1963) 431.
[3] A.W. Robinson, P. Gardner, A.P.J. Stampfl, R. Martin, G. Nyberg, J. Electron Spectrosc. Rel. Phenomena, in press.
[4] A.W. Robinson, P. Gardner, A.P.J. Stampfl, R. Martin, G. Nyberg, Surf. Sci. 387 (1997) 243.
[5] D.A. Shirley, Phys. Rev. B 5 (1972) 4709.
[6] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, The Art of Scientific Computation, 2nd ed. (Cambridge Univ. Press, Cambridge, 1992), ISBN 0-521043108-5.