

LSFR: Computing packages for data analysis

Jordan Hulme and Adam Petrus

Department Of Physics & Astronomy, The University Of Manchester,
Manchester M13 9PL, England

18th December, 2003

Abstract

We have developed two computer packages for data analysis, designed to match the needs of physics undergraduates. The first package is in the form of a script for MATLAB, while the second is a standalone program written in C++, with extended capabilities.

Both packages perform least-squares regression analysis on a user-supplied dataset, and calculate appropriate fitted parameters (with uncertainties) and an associated χ^2 value.

1 Aims & Objectives

The aim of this project was to create a new version of the MATLAB least-squares fit script available on the physics departmental network, together with appropriate documentation such as online help. The script is used in first-year laboratory work, as part of a course on data analysis, and students are encouraged to use it to analyse results of experiments.

As the script is to be used for teaching purposes, it needs to be fairly straightforward to use. Major concerns in this area is the user-friendliness of the interface, as well as the accuracy and stability of the program.

It was also decided to extend the capabilities of the script. In the original script, the only fit types possible were a simple linear fit, and a simple quadratic fit. The new script was required to add support for at least a linear fit through the origin. In addition, we have added support for a cubic fit.

2 Introduction

It was decided to rewrite the MATLAB script from scratch for two reasons. Firstly, the GUIDE development environment in MATLAB 6 allows a graphical user interface (GUI) to be produced fairly quickly and easily, and analysis code added later; secondly, because it is often difficult to understand and modify another person's code

MATLAB is a powerful mathematical package, with the capabilities to design

both command-line and GUI-based scripts. It has a large library of mathematical functions built in, reducing the amount of programming required (and also ensuring that calculations are carried out correctly and as efficiently as possible). There are also several predefined functions for loading and processing data from files, with built-in checks for invalid data. In contrast, the standard C++ mathematical libraries are fairly limited. Matrix and vector classes are not built-in, and so relevant classes had to be developed before the main analysis code could be written.

Although MATLAB is powerful, it has several limitations. The GUI development environment is fairly limited, and can be inefficient when working with related controls. More importantly, in order to use the MATLAB script version of *LSFR*, a user needs to have a copy of MATLAB available, which means either using it in the Physics departmental computer clusters, or buying a copy for themselves.

A standalone program would not have this requirement since it is a self-contained application, which only requires any recent version of Windows. This could be extremely useful to students; for example, if lab work needs to be analysed and written up at home over the Christmas holiday, students will not necessarily have access to MATLAB. The object-oriented nature of C++ also makes functions such as fitting multiple data sets to a single function much easier and simpler than the equivalent code in MATLAB would be to write.

3 Mathematical and Numerical Methods

The most important part of the packages developed is their capability to perform accurate analysis of a given dataset. This requires the calculation of the parameters appropriate to the fit (e.g. for a simple straight line, its slope and intercept), the uncertainties or errors associated with these parameters, and some measure of the goodness of the fit.

3.1 Calculation of fitted parameters

3.1.1 Simple linear fit ($y = mx + c$)

For any general data set, there exist a series of abscissae ($x_1, x_2, x_3, \dots, x_n$) and a corresponding series of ordinates ($y_1, y_2, y_3, \dots, y_n$). These can be plotted on a graph, and an estimate of the best-fitting line through the points can be made (figure 1). Each of the data points now has a deviation from this line associated

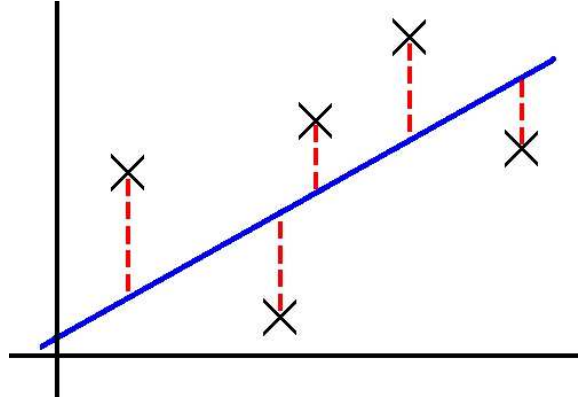


Figure 1: Deviations from a fitted line

with it, given by the formula

$$\delta_i = mx_i + c - y_i \quad (1)$$

The total deviation over all the points is then given by

$$\begin{aligned} G &= \sum_{i=1}^n (\delta_i)^2 \\ &= \sum_{i=1}^n (mx_i + c - y_i)^2 \end{aligned} \quad (2)$$

known as the *Goodness-of-Fit*. Note that the individual deviations are squared to ensure positive contributions to the sum. For the best fitting line, G is at a minimum. Taking a linear fit as an example, this means that

$$\begin{aligned} \frac{\partial G}{\partial m} &= 0 & \frac{\partial G}{\partial c} &= 0 \\ \frac{\partial G}{\partial m} &= \sum_i 2x_i (mx_i + c - y_i) & \frac{\partial G}{\partial c} &= \sum_i 2(mx_i + c - y_i) \\ \sum mx_i^2 + \sum cx_i - \sum x_i y_i &= 0 & \sum mx_i + \sum c - \sum y_i &= 0 \end{aligned}$$

Defining average values of the form

$$\langle x \rangle = \frac{1}{n} \sum_i x_i,$$

taking the constants out of the sums and dividing through by n gives

$$m \langle x \rangle + c = \langle y \rangle \quad m \langle x^2 \rangle + c \langle x \rangle = \langle xy \rangle$$

These are simultaneous equations in m and c , and the solutions are

$$m = \frac{\langle xy \rangle - \langle x \rangle \langle y \rangle}{\langle x^2 \rangle - \langle x \rangle \langle x \rangle} \quad (3)$$

$$c = \frac{\langle y \rangle \langle x^2 \rangle - \langle xy \rangle \langle x \rangle}{\langle x^2 \rangle - \langle x \rangle \langle x \rangle} \quad (4)$$

Note that the best-fitting line always passes through the point $(\langle x \rangle, \langle y \rangle)$.

3.1.2 Weighted linear fit

For data with associated error estimates $(\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_n)$, equation 2 is modified to give

$$G = \sum_{i=1}^n \left(\frac{\delta_i}{\sigma_i} \right)^2 \quad (5)$$

The value $\frac{1}{\sigma_i^2}$ is known as the weight, w_i , of the point (x_i, y_i) . It is calculated in this way so that points with a smaller estimated error are given a higher weighting in the calculation. In this case, when G is at a minimum, it is conventionally called χ^2

$$\chi^2 = \sum_{i=1}^n w_i \delta_i^2 \Big|_{\min} \quad (6)$$

Following the above procedure, it can be shown that when weights are included, the equations for m and c are

$$m = \frac{\langle w \rangle \langle wxy \rangle - \langle wx \rangle \langle wy \rangle}{\langle w \rangle \langle wx^2 \rangle - \langle wx \rangle \langle wx \rangle} \quad (7)$$

$$c = \frac{\langle wy \rangle \langle wx^2 \rangle - \langle wxy \rangle \langle wx \rangle}{\langle w \rangle \langle wx^2 \rangle - \langle wx \rangle \langle wx \rangle} \quad (8)$$

3.1.3 Straight line through origin ($y = mx$)

In this case, the deviation of each point is given by

$$\delta_i = mx_i - y_i$$

The value of χ^2 is therefore

$$\chi^2 = \sum_{i=1}^n w_i (mx_i - y_i)^2 \Big|_{\min}$$

Following the same procedure,

$$\begin{aligned} \frac{\partial \chi^2}{\partial m} &= \sum_i 2w_i x_i (mx_i - y_i) \\ &= m \sum w_i x_i^2 - \sum w_i x_i y_i \end{aligned}$$

$$\begin{aligned}
m \langle wx^2 \rangle - \langle wxy \rangle &= 0 \\
m &= \frac{\langle wxy \rangle}{\langle wx^2 \rangle}
\end{aligned} \tag{9}$$

3.1.4 Polynomial fit ($y = a_0 + a_1x + a_2x^2 + \dots$)

For any polynomial fit, the deviations are given by

$$\delta_i = (a_0 + a_1x_i + a_2x_i^2 + \dots - y_i)$$

Taking the standard equation for χ^2 ,

$$\chi^2 = \sum w_i \delta_i^2$$

Now

$$\begin{aligned}
\frac{\partial \chi^2}{\partial m} &= \sum 2w_i \delta_i \\
\frac{\partial \chi^2}{\partial a_1} &= \sum 2w_i x_i \delta_i \\
\frac{\partial \chi^2}{\partial a_2} &= \sum 2w_i x_i^2 \delta_i \\
&\vdots
\end{aligned}$$

Expanding out each of these equations gives

$$\begin{aligned}
\sum (a_0 w_i + a_1 w_i x_i + a_2 w_i x_i^2 + \dots - w_i y_i) &= 0 \\
\sum (a_1 w_i x_i + a_2 w_i x_i^2 + a_3 w_i x_i^3 + \dots - w_i x_i y_i) &= 0 \\
\sum (a_2 w_i x_i^2 + a_3 w_i x_i^3 + a_4 w_i x_i^4 + \dots - w_i x_i^2 y_i) &= 0 \\
&\vdots
\end{aligned}$$

In general, these linear simultaneous equations can be formed into a matrix equation. Using the same convention that

$$\langle x \rangle = \frac{1}{n} \sum_i x_i,$$

this gives

$$\begin{pmatrix} \langle w \rangle & \langle wx \rangle & \langle wx^2 \rangle & \dots \\ \langle wx \rangle & \langle wx^2 \rangle & \langle wx^3 \rangle & \dots \\ \langle wx^2 \rangle & \langle wx^3 \rangle & \langle wx^4 \rangle & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} \langle wy \rangle \\ \langle wxy \rangle \\ \langle wx^2 y \rangle \\ \vdots \end{pmatrix} \tag{10}$$

This equation can be solved by pre-multiplying both sides by the inverse of the $\langle wx^n \rangle$ matrix:

$$\mathbf{A}\mathbf{b} = \mathbf{c}$$

$$\mathbf{A}^{-1}\mathbf{A}\mathbf{b} = \mathbf{A}^{-1}\mathbf{c}$$

but $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$, the identity matrix, and so

$$\mathbf{b} = \mathbf{A}^{-1}\mathbf{c}$$

The square matrix created is of order $(n \times n)$, where n is the order of the polynomial involved. For a polynomial of order 1 (i.e. a straight line), equation 10 solves to produce the same expressions for m ($= a_1$) and c ($= a_0$) given in equations 3 and 4.

3.2 Errors on fitted parameters

The general formula for the error associated with a function $F(x, y, z)$ of other variables with associated errors is

$$\sigma_F^2 = \left(\frac{\partial F}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial F}{\partial y}\right)^2 \sigma_y^2 + \left(\frac{\partial F}{\partial z}\right)^2 \sigma_z^2 \quad (11)$$

Since the only variable to have an associated error in the parameter calculations is the dependent variable, y_i , the error on a given fitted parameter a is therefore

$$\sigma_a^2 = \sum_i \left(\frac{\partial a}{\partial y_i}\right)^2 \sigma_i^2 \quad (12)$$

3.2.1 Linear fit through origin

$$\begin{aligned} \frac{\partial m}{\partial y_i} &= \frac{1}{\langle wx^2 \rangle} \frac{\partial}{\partial y_i} \langle wxy \rangle \\ &= \frac{1}{n} \frac{w_i x_i}{\langle wx^2 \rangle} \end{aligned}$$

Note that $n \langle wx^2 \rangle = \sum wx^2$. The error associated with m is therefore

$$\sigma_m^2 = \sum_i \left(\frac{w_i x_i}{\sum wx^2}\right)^2 \sigma_i^2 \quad (13)$$

However, recall that $w_i = \frac{1}{\sigma_i^2}$. Substituting this back into equation 13, and squaring out the bracket gives

$$\begin{aligned} \sigma_m^2 &= \frac{\sum wx^2}{(\sum wx^2)^2} \\ \sigma_m^2 &= \frac{1}{\sum wx^2} \end{aligned} \quad (14)$$

3.2.2 Linear fit

$$\frac{\partial m}{\partial y_i} = \frac{1}{\langle w \rangle \langle wx^2 \rangle - \langle wx \rangle \langle wx \rangle} \cdot \frac{\partial}{\partial y_i} \left(\langle w \rangle \frac{1}{n} \sum_i w_j x_i y_j - \langle wx \rangle \frac{1}{n} \sum_i w_i y_i \right)$$

In each of the sums, only one term contains y_i and is nonzero. All other terms cancel, leaving

$$\frac{\partial m}{\partial y_i} = \frac{\langle w \rangle \frac{1}{n} w_i x_i - \langle wx \rangle \frac{1}{n} w_i}{\langle w \rangle \langle wx^2 \rangle - \langle wx \rangle \langle wx \rangle} \quad (15)$$

Similarly,

$$\frac{\partial c}{\partial y_i} = \frac{\langle wx^2 \rangle \frac{1}{n} w_i - \langle wx \rangle \frac{1}{n} w_i x_i}{\langle w \rangle \langle wx^2 \rangle - \langle wx \rangle \langle wx \rangle} \quad (16)$$

and as above,

$$\sigma_m^2 = \sum_i \left(\frac{\partial m}{\partial y_i} \right)^2 \sigma_i^2$$

$$\sigma_c^2 = \sum_i \left(\frac{\partial c}{\partial y_i} \right)^2 \sigma_i^2$$

3.2.3 Polynomial fit

The errors associated with polynomial parameters can be found using a similar matrix equation to [10](#). The only y -dependence in the equation is in the right-hand column vector. Applying equation [11](#) to this vector gives a corresponding error vector

$$\begin{pmatrix} \sum \left(\frac{\partial}{\partial y_i} \langle wy \rangle \right)^2 \sigma_i^2 \\ \sum \left(\frac{\partial}{\partial y_i} \langle wxy \rangle \right)^2 \sigma_i^2 \\ \sum \left(\frac{\partial}{\partial y_i} \langle wx^2 y \rangle \right)^2 \sigma_i^2 \\ \vdots \end{pmatrix} = \begin{pmatrix} \sum \left(\frac{w_i}{n} \right)^2 \sigma_i^2 \\ \sum \left(\frac{w_i x_i}{n} \right)^2 \sigma_i^2 \\ \sum \left(\frac{w_i x_i^2}{n} \right)^2 \sigma_i^2 \\ \vdots \end{pmatrix} \quad (17)$$

The error values can now be found by pre-multiplying by the same matrix \mathbf{A} , as before, giving

$$\begin{pmatrix} \sigma_{a_0}^2 \\ \sigma_{a_1}^2 \\ \sigma_{a_2}^2 \\ \vdots \end{pmatrix} = \sum_i \left[\mathbf{A}^{-1} \begin{pmatrix} \left(\frac{w_i}{n} \right) \\ \left(\frac{w_i x_i}{n} \right) \\ \left(\frac{w_i x_i^2}{n} \right) \\ \vdots \end{pmatrix} \right]^2 \sigma_i^2 \quad (18)$$

Again, for a polynomial of order 1, the expression in the square brackets produces equations [15](#) and [16](#) derived above.

3.3 Fitting several straight lines with different intercepts but the same slope

This takes a very similar form to the simple linear fit. We start with a slightly modified form of the the goodness-of-fit equation for a straight line:

$$\chi^2 = \sum_{j=1}^p \sum_{k=1}^n w_{jk} (mx_{jk} + c_j - y_{jk})^2 \quad (19)$$

where j is the set number for a series of p sets of data, each with a different intercept. Again we calculate the partial derivatives of χ^2 with respect to each coefficient:

$$\begin{aligned} \frac{\partial \chi^2}{\partial m} &= 2 \sum_{j=1}^p \sum_{k=1}^n w_{jk} (mx_{jk} + c_j - y_{jk}) x_{jk} \\ \frac{\partial \chi^2}{\partial c_j} &= 2 \sum_{k=1}^n w_{jk} (mx_{jk} + c_j - y_{jk}) \end{aligned}$$

With these partial derivatives set to zero, this leaves $(p+1)$ simultaneous equations of the below form:

$$\begin{aligned} m \sum_{j=1}^p \sum_{k=1}^n w_{jk} x_{jk}^2 + \sum_{j=1}^p c_j \sum_{k=1}^n w_{jk} x_{jk} &= \sum_{j=1}^p \sum_{k=1}^n w_{jk} x_{jk} y_{jk} \quad (20) \\ m \sum_{k=1}^n w_{1k} x_{1k} + c_1 \sum_{k=1}^n w_{1k} &= \sum_{k=1}^n w_{1k} y_{1k} \\ m \sum_{k=1}^n w_{2k} x_{2k} + c_2 \sum_{k=1}^n w_{2k} &= \sum_{k=1}^n w_{2k} y_{2k} \\ &\vdots \\ m \sum_{k=1}^n w_{pk} x_{pk} + c_p \sum_{k=1}^n w_{pk} &= \sum_{k=1}^n w_{pk} y_{pk} \quad (21) \end{aligned}$$

As before, this can be written in the form of a matrix¹, and so can be solved in the same way as equation 10.

$$\begin{aligned} &\begin{pmatrix} \sum_j \sum_k w_{jk} x_{jk}^2 & c_1 \sum_k w_{jk} x_{jk} & \cdots & \cdots & c_p \sum_k w_{jk} x_{jk} \\ \sum_k w_{1k} x_{1k} & \sum_k w_{1k} & 0 & \cdots & 0 \\ \sum_k w_{2k} x_{2k} & 0 & \sum_k w_{1k} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_k w_{pk} x_{pk} & 0 & 0 & \cdots & \sum_k w_{pk} \end{pmatrix} \begin{pmatrix} m \\ c_1 \\ c_2 \\ \vdots \\ c_p \end{pmatrix} \\ &= \begin{pmatrix} \sum_j \sum_k w_{jk} x_{jk} y_{jk} \\ \sum_k w_{1k} \\ \sum_k w_{2k} \\ \vdots \\ \sum_k w_{pk} \end{pmatrix} \quad (22) \end{aligned}$$

¹**Note:** For simplicity, the sums in equation 22 have been abbreviated. In each case, a sum over j is from $1 \rightarrow p$, and a sum over k is from $1 \rightarrow n$.

Both programs use the matrix-based calculations derived above for polynomial fitting, as this allows calculations to be both initialised and performed quickly and efficiently. (The calculations for $y = mx$ are short and simple enough to be coded explicitly.) This also reduces the amount of code required, as since any general polynomial fit of order m will produce an m -by- m matrix of the form shown in 10, the same code can be used to populate and solve the matrix equation using a simple loop as shown below, simply by specifying the order of the polynomial.

```

for I = 0:m
  for J = 0:m
    A(I,J) = EwxP(x, w, I+J)/N;
  end
  B(I) = EwyxP(x, y, w, I)/N;
end

```

where the function ‘EwxP’ performs the calculation

$$\sum_{i=1}^N w_i x_i^P$$

using the specified data sets x and w , and power P (and similarly for ‘EwyxP’).

3.4 Interpretation of χ^2

Once calculations have been performed, and a value for χ^2 calculated, this value can then be compared against a table of critical values of the χ^2 *distribution*. These critical values show the statistical probability of a particular value of χ^2 occurring. The comparison allows a judgment to be made about the data, and in particular whether the values of the dependent variable y are normally distributed, given the fit type and error estimates provided. An incorrect fit type, or over- or underestimated errors will produce a ‘bad’ value of χ^2 .

3.5 Problems with Numerical Computing

Most computers use floating point representation to symbolize a real number in decimal form, by shifting the decimal point and supplying appropriate powers of 10. Figure 2 shows the set of standardised methods for storing numbers on a system. MATLAB uses the ‘double’ type to store numbers, meaning it works to accuracy between 15 and 16 digits of precision. The C++ version of *LSFR* uses mostly the double type, and when suitable the float type. In addition it also uses the long double type for calculations that may suffer particularly badly from any loss of significance.

Certain numerical operations can cause loss of significance; for example, the subtraction of a value from a value very similar to it. Consider the function

$$y = \sqrt{x^2 + 1} - 1 \tag{23}$$

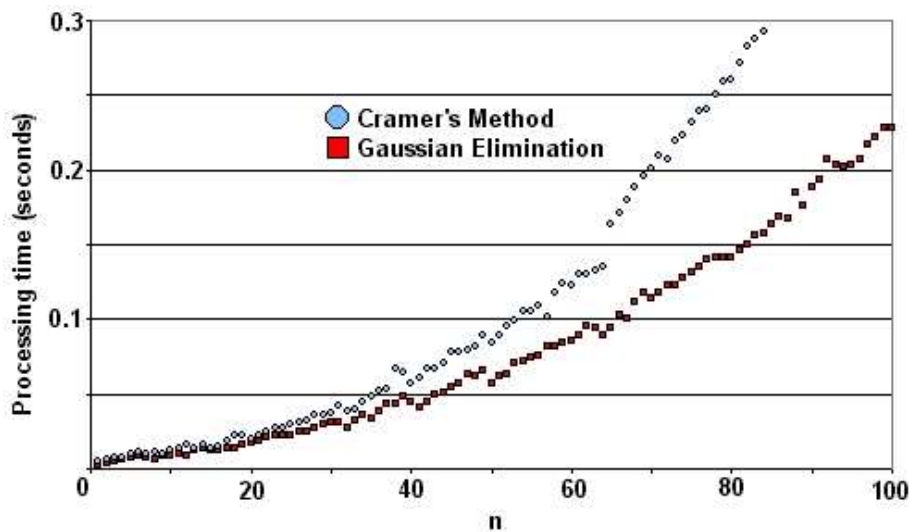


Figure 3: Processing time taken to solve matrices by Cramer's method and Gaussian elimination

3.5.1 Data scaling in \mathcal{LSFR}

In order to reduce loss of significance in the polynomial analysis routine, the data are scaled before any calculations are performed. (No scaling is applied for a fit of type $y = mx$.) Firstly, the mean of the data is subtracted from each point, changing the variable from x to $(x - \bar{x})$. Next, the order of magnitude of the data range is calculated, and the data scaled to have a mean order of unity. The analysis calculations are then performed with the scaled data. This also reduces the risk of 'overflow' in the calculations—that is, when the numbers stored are larger (or smaller) than the data type can hold—for significantly large- or small-magnitude data.

Once the fitted parameters have been found, the values must be scaled back up. The above process is reversed—the order of magnitude is restored, and then the original variable x is restored by performing a binomial expansion:

$$a_n (x - b)^n = \sum_{k=0}^n \binom{n}{k} a_n x^{n-k} b^k$$

The extra terms produced by this expansion are then added on to the calculated parameters to complete the routine.

3.6 Non-linear equations

Suppose we wish to fit a trendline which after least squares analysis leads to a set of nonlinear equations to solve. To illustrate consider a fit of the form

$$y = A \cos(Bx)$$

The parameter B enters the function in a nonlinear way creating some diffi-

culty. Applying the standard goodness-of-fit equation gives

$$\chi^2 = \sum_{k=0}^n w_k (y_k - A \cos B x_k)^2$$

Proceeding as before, the minimum of χ^2 is found by setting the derivatives of this equation to zero.

$$f_1(A, B) = \frac{\partial G}{\partial A} = -2 \sum_{k=1}^n w_k (y_k - A \cos B x_k) \cos B x_k \quad (25)$$

$$f_2(A, B) = \frac{\partial G}{\partial B} = 2 \sum_{k=1}^n w_k (y_k - A \cos B x_k) A x_k \sin B x_k \quad (26)$$

For such a problem, one can use Newton's method, which can be accomplished by expanding both functions via a Taylor expansion about $f(A_1, B_1)$, discarding higher-order terms and setting the functions to zero.

$$\begin{aligned} f_1(A_i + \delta A_i, B_i + \delta B_i) &= f_1(A_i, B_i) + \delta A_i \frac{\partial f_1}{\partial A_i}(A_i, B_i) + \delta B_i \frac{\partial f_1}{\partial B_i}(A_i, B_i) + \dots \\ f_2(A_i + \delta A_i, B_i + \delta B_i) &= f_2(A_i, B_i) + \delta A_i \frac{\partial f_2}{\partial A_i}(A_i, B_i) + \delta B_i \frac{\partial f_2}{\partial B_i}(A_i, B_i) + \dots \end{aligned}$$

This results in two linear equations for δA_1 and δA_2 , which can be written in the form of a matrix:

$$\begin{pmatrix} \frac{\partial f_1}{\partial A} & \frac{\partial f_1}{\partial B} \\ \frac{\partial f_2}{\partial A} & \frac{\partial f_2}{\partial B} \end{pmatrix} \begin{pmatrix} \delta A_i \\ \delta B_i \end{pmatrix} = - \begin{pmatrix} f_1(A_i, B_i) \\ f_2(A_i, B_i) \end{pmatrix} \quad (27)$$

Thus an iterative method can be used if we define $A_{i+1} = A_i + \delta A_i$ and $B_{i+1} = B_i + \delta B_i$, and argue that these should produce improved approximations to the solutions if the higher order terms are sufficiently small to be neglected. This assumption is fair as long as good initial starting values are provided. Otherwise, consecutive iterations may diverge from the true root.

Newtons method can be utilized for more than just two equations. It can apply to any number of sets of nonlinear equations. Newtons Method for a system of N equations may be written as

$$Q_{n+1}^{(n)} = Q_m^{(n)} - \left(J_m^{(n)} \right)^{-1} F_m^{(n)}$$

where F is the vectorised set of functions f_1, \dots, f_n , Q is the vectorised solutions, $J_m^{(n)}$ is an n-by-n Jacobian matrix and its (i, j) component is defined as

$$J_{i,j}^{(n)} = \frac{\partial f_i}{\partial q_j}, i, j = 1 : n$$

In practice, Newtons method has two main drawbacks. Obtaining good starting values for A and B can be difficult or sometimes even impossible. Also, it is generally too expensive when N is large. An approach to this is to avoid

directly calculating the Jacobian by substituting finite differences, as illustrated in equation 28.

$$J_{i,j}^{(n)} \approx \frac{f_i(q_{1,n}, \dots, q_{n,m}) - f_i(q_{1,m}, \dots, q_j + h, \dots, q_{n,m})}{h} \quad (28)$$

H can be chosen as perturbation to the jth component of Q, giving

$$J_{i,j}^{(n)} \approx \frac{f_i(q_{1,n}, \dots, q_{n,m}) - f_i(q_{1,m}, \dots, q_{j,m-1}, \dots, q_{n,m})}{q_{j,m} - q_{j,m-1}}$$

Nonlinear analysis is not enabled in \mathcal{LSFR} by default, but some sample nonlinear fit routines can be enabled in the standalone program from the preferences menu by ticking the ‘enable samples’ checkbox.

3.7 The custom-fit plugin

As it stands, \mathcal{LSFR} contains support only for simple polynomial functions. There are many functions other than an experienced undergraduate or postgrad would benefit from, but these were left out of \mathcal{LSFR} due to its primary role as a teaching aid for new undergraduates.

MATLAB is a fairly easy interpretation language to learn and use, so adding custom routines to the \mathcal{LSFR} script is straight forward with intermediate programming knowledge and experience. However, modifying the C++ version requires more computing knowledge, and much of the large amount of existing code is irrelevant to the task of writing a fit routine.

A way to help the more astute student to complete this task is to hide the low level details that make up the guts of the program (GUI and graph plotting) and direct them to the areas that are relevant to the task. To achieve this relevant code has been placed in a separate C++ project, compiled separately from the main program which reduces accidental modification to the program and significantly reduces the compile time.

The resulting separate C++ project has just eight functions to modify to accomplish a successful custom fit routine. These functions are:

- customfitname()** a function to add the name of the fit to the available options.
- customispoly()** a function to determine if the custom fit is just a general polynomial which returns its order. If this is the case, then the remaining functions are redundant as the standard function is used.
- customestimates()** a function to return the amount of estimates/initial values to perform the fit.
- customfit::equ(double x)** a function to represent the equation of the fit.
- customfit::fitl(bool high)** a function to perform least squares fitting of the routine.
- customfit::genstrs(TStrings *S, bool on)** a function to generate the text to display the calculated parameters.
- customfit::func(int ii)** a different representation of the equation of the fit, used when exporting to gnuplot.

In addition, a step-by-step tutorial for four examples has been added to the help documentation.

4 Development of the program and GUI

4.1 User-Friendliness

Since the packages are to be used as teaching aids, it is important that they are easy to use. The old fit package suffers from several user-friendliness issues. Firstly, the GUI is minimal—it consists only of a series of buttons in a window. Clicking a button either changes an option or takes the user to the MATLAB command window (e.g. to type in a new plot title). *LSFR* makes use of the capabilities of MATLAB to add ‘uicontrol’ objects (buttons, edit boxes, checkboxes etc.) to a figure window. This has allowed us to create a user interface that is self-contained, and which has a form similar to that used in standard Windows-based software.

In addition, the labelling of the buttons in the old package is somewhat unclear, particularly the button used to select the fit type. This button displays the fit type that will be selected if you click the button, but the current fit type is not displayed anywhere in the GUI.

4.2 Designing the front-end

There are many programs available that can be used for data analysis and graph plotting

In order to make *LSFR* as accessible as possible, it was decided that the GUI should conform to standard WIMP (Windows, Icons, Menus and Pointers) environment conventions—for example, a list of mutually-exclusive options is indicated by a drop-down menu or a series of radio buttons, or a variable can be changed directly by using an edit box. Controls that perform similar or related functions are grouped together by using frames, and controls can be navigated in a logical order by using the tab key.

The implementation of WIMP conventions in MATLAB is limited by the relatively minimal GUI support—the GUIDE system is a relatively new addition to MATLAB, and only the most basic controls are supported, often to a limited degree. For example, whereas in a dedicated GUI development environment such as Borland C++ Builder it is possible to specify that certain controls are mutually exclusive, in MATLAB the corresponding code must be written manually. In addition, future versions of MATLAB may not necessarily be backwardly compatible with GUI’s created in earlier versions, so this was another factor in deciding to write a standalone routine.

4.2.1 MATLAB version (figure 4)

To keep the interface as simple as possible, the GUI for the MATLAB version is designed as a single window, with options gathered together in logical groups. All options are represented by controls on this window, rather than using menus and pop-up dialogs. Figures are produced in their own windows, using MATLAB’s plotting functions to do so. The standard MATLAB figure window menu has been removed from the *LSFR* figure windows, and a simplified menu with only basic options added in its place. This limits the user to a few simple tasks (saving, printing and copying), and removes the more advanced options for figure manipulation—from a teaching point of view this is ideal, as it means that figures can be produced consistently.

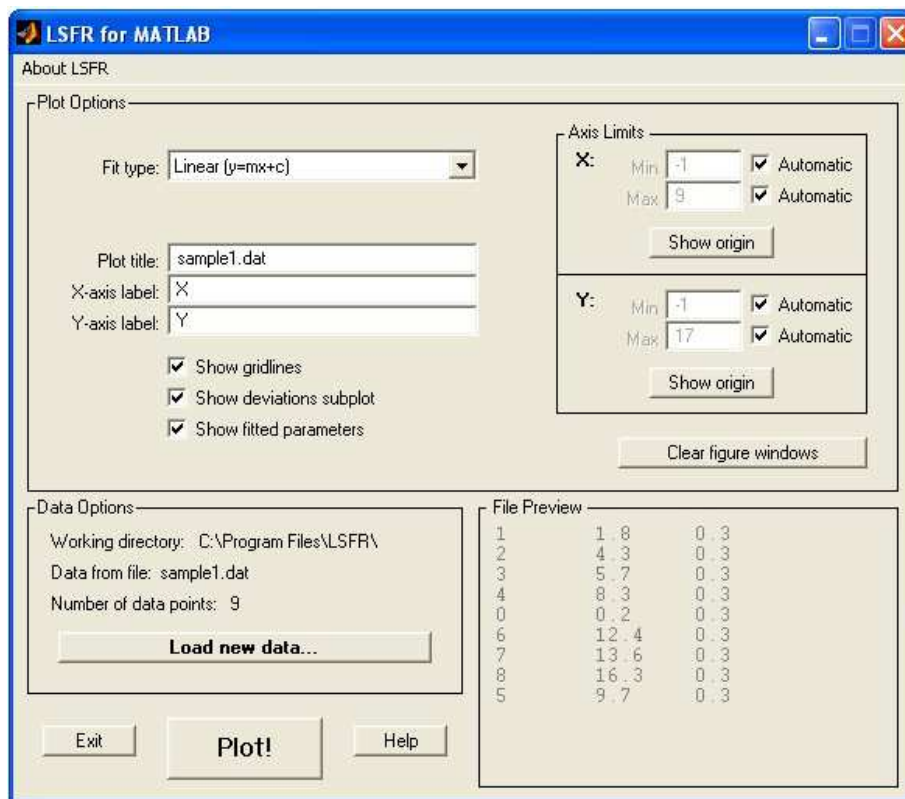


Figure 4: GUI for \mathcal{LSFR} (MATLAB version)

4.2.2 Standalone version (figure 5)

The standalone was developed under Borland C++ Builder, allowing the user interface to be built relatively quickly and easily. It has a multiple document interface (MDI), allowing multiple ‘child’ windows to be created within the main *LSFR* window. There are four specific child window types. These are:

- The logbook displays messages sent from other children windows, and can be used for debugging purposes. Only one instance of the window can be generated.
- The Scatchpad child contains a basic text editor, allowing data to be entered or loaded from a file.
- The charting window is formed when a data from an instance of scatchpad is converted to a chart. It has four tab pages. The first is for importing data, the second allows the viewing of a dataset, the third displays the fit-type options, and the final page contains the plot of the data and a subplot of the deviations from the fitted trendlines in a similar way to the MATLAB version.
- The Teaching child initialises an instance of Internet Explorer, allowing the user to view the Physics teaching pages from within the program.

The main parent window of application contains the menu system and toolbar, and is responsible for sending messages to children when the main menu, and toolbar items are clicked.

4.3 Producing graphs

For the MATLAB version of *LSFR*, producing the graph is relatively straightforward: MATLAB is a mathematical computing package, and so contains all the necessary commands to both plot graphs and customise the display as required. For each graph plotted, a new figure window is produced containing the main graph and (depending on the options selected) a subplot of the deviations of the data points from the fitted line, together with a display of the calculated parameters and χ^2 value (figure 6). By default, *LSFR* automatically calculates appropriate limits for the axes, such that the entire data set is displayed. If these limits are not ideal, limits can be specified manually on the main window before plotting. Any invalid limits (for example, a maximum less than a minimum, or a range that does not include any data points) produce a warning dialog, and *LSFR* will not attempt to plot a graph until valid limits are set.

Borlands C++ Builder 5 includes a version of Steams Softwares charting libraries (TeeChart). This allows charts to be created just by providing it with the data and other details such as axis increments and titles, and calculates variables such as appropriate axis limits automatically. However this basic and early version of TeeChart did not include any error bar plotting series. To add this functionality, we developed our own error bar series, located in the `APErrorSeries.cpp/.h` files.

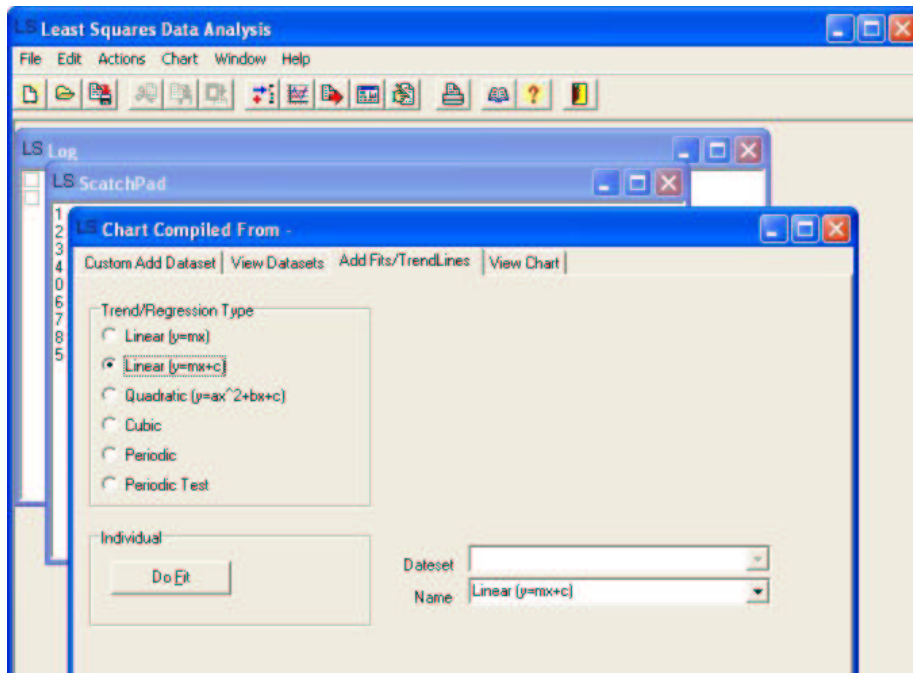


Figure 5: GUI for \mathcal{LSFR} (standalone version)

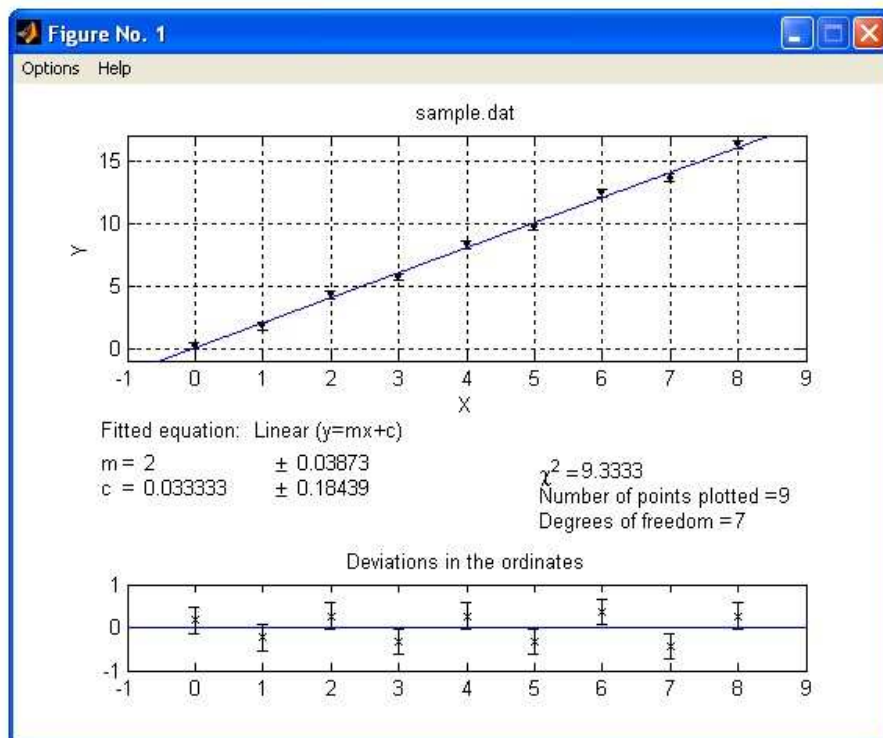


Figure 6: The \mathcal{LSFR} figure window in MATLAB

5 Conclusions

Developing the two versions simultaneously resulted in a large increase in the amount of programming that needed to be done. However, in terms of the actual analysis code (rather than the GUI) it actually reduced the workload, as once a routine was successfully running in one program, it could be relatively easily ported across to the other.

It was also useful to have the existing MATLAB script available, as this enabled us to check that our calculations were producing the correct results by processing the same data file with each program.

In order to test the ease-of-use and reliability of the program, it was subject to several ‘beta’ releases before the final version. The first beta of the MATLAB script was tested by selected first-year students in November, and the feedback from this allowed us to fix several bugs that were found, in addition to refining the interface.

The second beta release, together with a beta release of the standalone program, was made available to all students through the departmental network in December, and feedback from this has been mostly positive.

The user-friendliness of the interface has been greatly improved by developing a full GUI, and the code itself is both more stable (in terms of dealing with invalid or badly-scaled data) and more efficient in calculations and memory usage than the original MATLAB script. The extended fitting capabilities—in particular, the $y = mx$ fit—also make it useful for a wider range of experimental results. Logarithmic and exponential fitting routines were considered, but it was decided not to include them since a function of this form can easily be converted into a linear equation, and solved using the existing code.

Several items of documentation have also been produced, including short quick-start guides for both versions, a more detailed user’s manual for the MATLAB script and a Windows Help file for the standalone.

Both of the programs are now ready to release, and have been tested rigorously to remove as many bugs or errors as possible. The files will now be uploaded to the physics network for the use of all students, and also for teaching purposes.

References

- [1] Borland C++ Builder documentation (Borland, 2003)