Using Matlab for Curve Fitting in Junior Lab

MIT Department of Physics
Advanced Experimental Physics I & II
(Dated: August 31, 2004)

1. INTRODUCTION

Curve fitting is one of the most common analytical tasks you will perform during Junior Lab. Students are welcome to utilize any set of routines for curve fitting as long as the standards for reporting results, identified in this and other Junior Lab documents, are met.

This short guide is designed to get you started using Matlab, a commercial product available on both Athena and the Windows PCs in Junior Lab. Academic versions of Matlab are also relatively inexpensive for those students wishing to run it on their own computers. While Matlab is the default Junior Lab solution, some students prefer to use alternative mathematics packages (e.g. Maple, Mathematica or LabVIEW). The only requirement is a solid understanding of the underlying algorithm and mathematics, detailed in many places, most notably the required Junior Lab text, see [1].

2. STARTING MATLAB

Using Matlab on Athena[1] requires the following steps:

1. From the Athena prompt, attach the Matlab locker (you can also add this to you .cshrc.mine file to have it automatically attached at login):

```
% add matlab
```

2. Open Matlab into a new window:

```
% matlab &
```

Once it has finished loading, it will present you with the Matlab prompt: `>>`. From this prompt you can execute an of the Matlab commands or run a Matlab script. To run a script, first make sure it ends in `.m` and resides in your `matlab` directory and then simply type the name at the prompt (without the `.m`):

```
>> myscript
```

[1] These specific instructions are for the Athena installation of Matlab. Very similar (if not identical) operations will work on Windows platforms as well.

3. USING MATLAB SCRIPTS

One very powerful and very easy way to using Matlab is to use scripts. Scripts are simply text files that contain a series of Matlab commands. The entire process of fitting will require at least a handful of commands so it is useful to have them all in a single script. Once you have this script you can return to it later, repeat your fit, make modifications, etc. without having to retype all of the commands.

If you plan to use Matlab scripts, it's a good idea to create a 'matlab' directory in your home directory. You can do this by typing:

```
% cd ~; mkdir matlab
```

You should save any Matlab scripts that you write in this directory. You can use emacs (or any other text editor) to create and edit Matlab scripts. For example, to create a new script called `myscript.m` type:

```
% emacs myscript.m &
```

Notice the "`.m`" extension on the script. All Matlab scripts must end in "`.m`" in order to execute correctly.

4. USING THE JUNIOR LAB FITTING TEMPLATE SCRIPT

To accommodate quick and easy fitting for Matlab beginners, we have created a script that you can use as a template for fitting. This script is available on the Junior Lab homepage or you can copy it (and a sample data set upon which to perform a sample fit) directly into you're `matlab` directory by typing

```
% cp /mit/8.13/matlab/fittemplate.m ~/matlab
% cp /mit/8.13/matlab/gauss3.dat ~/matlab
```

Before learning how to manipulate the script for your own use, try running it by typing (within Matlab):

```
>> fittemplate
```

As you can see, a Matlab scripts are is a very powerful tool with which you will want to become very familiar!

The basic procedure for using this fitting script is outlined as follows:

1. **Open** the script in Emacs.

2. **Modify** the script such that it:

(a) Loads data from your file into the vectors `x` and `y`

(b) Assigns the appropriate errors

(c) Contains the functional form that you want to fit

(d) Uses the options needed for your fit (starting point, upper and lower bounds, linear/nonlinear, etc.)

(e) Plots figures with labels appropriate for your data

3. **Save** the file with a new name, keeping the .m extension

4. Open **Matlab** and run the script

5. Matlab will **fit** your data, output the information relevant to the fit, and **plot** (1) the fitted curve on top of the original data and (2) the residuals.

This process will be outlined in detail below in a fit to a sample data set.

4.1. Non-Linear Least Squares Example

In developing this guide, we have used the Statistics Reference Dataset "Gauss3" available from the National Institute of Standards in Technology at `http://www.itl.nist.gov/div898/strd/nls/data/gauss3.shtml`. Since NIST has kindly provided certified values for this dataset, it is an excellent test case for checking the acceptability of alternative data fitting products.

The sample dataset, entitled "gauss3.dat" and residing in the `http://web.mit.edu/8.13/matlab` directory, consists of two poorly resolved Gaussian peaks on a decaying exponential background and must be fit using using a general (nonlinear) custom model. It has normally distributed zero-mean noise with a variance of 6.25.

Here we step through the process of modifying the template script to fit this data. If you have not already done so, create a `matlab` directory and copy the template script into this directory (see above). Note also that the any and all parts of the "template script" may be entered directly from the matlab command line. This is useful when diagnosing script generated errors.

1. **Open the script in Emacs**

```
% emacs ~/matlab/fittemplate.m &
```

2. **Modify the script such that it:**

(a) Loads data from your file into the vectors `x` and `y`.

Currently the script is set up to load the simulated data file called 'gausse3.dat' described earlier. It can be changed to load whatever dataset name you require:

```
load gauss3;
```

The result of a `load` command on a tab-delimited text file is a single "matrix" variable with the name gauss3 (250 rows x 2 col). The next lines in the script assign values to the `x` and `y` vectors from this parent matrix.

```
x=gauss3(:,1);
y=gauss3(:,2);
```

(Note that this syntax utilizes the entire data set in the vector definitions. By placing indices on either side of the colon, a subset of the entire data file may be selected for fitting.)

(b) Assigns the appropriate errors

Cftool requires that you specify the weights, w_i, for doing a fit. The weights are simply given by

$$w_i = 1/\sigma_i^2. \tag{1}$$

In the case of gauss3, we have a constant variance of 6.25 so we can assign the weights with the commands:

```
sig=ones(size(x))*sqrt(6.25);
wgts=1./sig.^2;
```

which creates a weights vector of the same size as the vector `x` with a constant value of one over 6.25.

Note: Since there are several different ways you could assign values to the error vector `sig`, the script includes several different assignment statements that cover a few cases. Once you have chosen one and modified it for the fit you are doing, you will need to *comment out* the other lines that assign values to sigma. To comment something out, simply put a '`%`' in front of it. Latex regards any thing following a '`%`' as a comment and does not interpret it. You'll notice that the template is heavily commented to explain what the commands are doing.

(c) Contains the functional form that you want to fit

The script currently contains several functional forms which you might find useful (Linear, Gaussian, Exponential, and Lorentzian). Since we are trying to fit the sum of two Gaussian s and a decaying exponential we'll need to create a custom equation for the fit. We can set up the fit model with the following syntax:

```
model = fittype('a*exp(-b*x)+a1*exp(-((x-
        b1)/c1)^2)+a2*exp(-((x-b2)/c2)^2)');
```

Then be sure to comment out the other fitting models so that only one is read by Latex. In general, if you need to fit a function that does not appear in the script, comment out all the functions that do appear and create a new one with the function that you need. Be sure to use `x` as the independent variable.

(d) Uses the options needed for your fit (starting point, upper and lower bounds, linear/nonlinear, etc.)

First you will need to decide if the fit you are doing is linear or nonlinear. (Keep in mind that just because you are fitting a nonlinear equation doesn't mean that you are doing a nonlinear fit. The fit is only nonlinear if the coefficients that you are fitting for appear nonlinearly in the equation).

If your fit is nonlinear, comment out the line that sets `'Value'` to `'LinearLeastSquares'`. If your fit is linear, comment out the line that sets `'Value'` to `'NonlinearLeastSquares'`.

If you are performing a nonlinear fit, your fit is much more likely to be successful if you specify a reasonable starting point and reasonable bounds. You should be able to determine such parameters by looking at a graph of your data. [2]

The variable `'opts.StartPoint'` will contain a vector with the starting values for your parameters. They should be assigned in alphabetical order. For the gauss3 data set, we have the following parameters:

```
opts.StartPoint=[100 100 40 0 100 150 20 20];
```

The variables opts.Lower and opts.Upper will contain vectors with the lower and upper bounds of the parameters respectively. These also need to be assigned in alphabetical order. If you decide to use these variables, please be sure to uncomment them by removing the % that is currently in front of them.

(e) Plots figures with labels appropriate for your data

Running the script will automatically generate plots for you. You'll need to make sure these graphs display the data you want with the correct labels.

Currently the script will generate one figure that is split into two sections. The top section will contain the fitted curve plotted on top of your data points with errorbars. The bottom section will contain the residuals of your fit.

The commands `'title'`, `'xlabel'`, and `'ylabel'` label the title, x axis and y-axis of the graph respectively. Right now the script just gives them generic titles. Make sure to modify these lines to give your graph meaningful labels with units.

3. **Save the file with a new name, keeping the .m extension.**
Be sure to save it with a *different* filename so that the template script will be left intact for your next fitting adventure.

4. **Open Matlab and run the script.**
If the script is saved in your matlab directory, you should be able to run it simply by typing in the name at the Matlab prompt (without the the `.m`). You may need to change directories if the file is stored elsewhere. Matlab has the same directory navigation commands as Athena such as `'ls'`, `'cd'`, `'pwd'`, etc.

5. **Matlab will fit your data, output the information relevant to the fit, and plot (1) the fitted curve on top of the original data and (2) the residuals.**

The script is currently setup to output some useful quantities that characterize your fit. These variables will be displayed in the Matlab window. They script causes these particular values to be output because they are declared with no semi-colon. In general, a semi-colon at the end of a Matlab command suppresses the output of the command.

The results of the fit are stored in `'fresult'`, `'gof'`, and `'output'`:

- `fresult` contains the coefficients for the fit with the confidence bounds.[3]
- `gof` contains quantities describing the goodness of the fit.[4]
- `output` contains other information such as the residuals and the Jacobian

To access any of the quantities stored within these three variables, use notation of the form `'variable.quantity'`. For example, you can get the Jacobian which is stored in `output` by typing

```
>> output.Jacobian
```

If there are other quantities related to the fit that you need to access, you can do so by simply typing in the name of the variable at the Matlab command line. For a list of the variables currently in Matlab's memory, use the command `'whos'`.

5. GENERATING PRESENTATION GRAPHICS

Figure 1 demonstrates a typical graphic you might create for an oral presentation or written summary. The

[2] Cftool has several fitting algorithms at its disposal. The most powerful fitting technique is the "Trust-Region" discussed in [4] which is the default non-linear method. It permits the use of lower and upper bounds around the initial guesses for the fitting parameters. The second method "Levenberg-Marquardt" is the one discussed in [1] and originally presented in [2, 3]. This method, is only slightly less powerful in that it cannot accept bounds on the fit parameters (though it does require good initial guesses). Try typing: **help fitoptions** to learn more about other fit options that you can select.

[3] Strictly speaking, the confidence bounds for fitted coefficients are given by: $C = b \pm t\sqrt{S}$ where b are the coefficients of the fit, t is the inverse of the Student's t cumulative distribution function (see Bevington, Appendix C.6 for more details on the origin and significance of the Student t distribution or try `'help tinv'` from inside Matlab), and S is a vector of the diagonal elements of the covariance matrix of the coefficient estimates, $(X^T X)^{-1} s^2$. Here $s^2 = \chi^2_{\nu-1}$. Refer to "Linear Least Squares" from Matlab's online documentation or Reference [1] for more information about X and X^T.

[4] The Matlab toolbox returns a goodness-of-fit statistic called SSE (Sum of Squares due to Error). This is simply the χ^2 statistic used frequently by physicists and in Bevington. The other necessary statistics is DFE (Degrees of Freedom) and is equal to the number of data points minus the number of fitted coefficients. In Junior Lab, we are most often interested in the reduced chi-square ($\chi^2_{\nu-1}$) which is simply given by the χ^2 divided by the DFE.

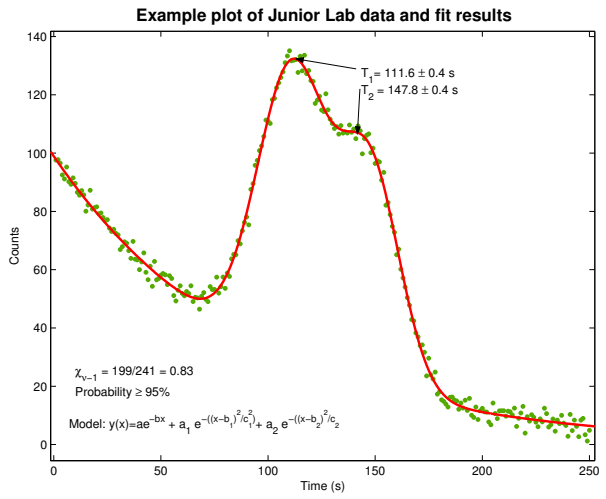

FIG. 1: This is an example of a basic figure for Junior Lab notebooks, presentations and written summaries. The dataset consists of two poorly resolved gaussian peaks on an exponential background. The noise is gaussian distributed with zero-mean and a variance of 6.25. **You should check with your individual section instructor for explicit instructions on how to prepare figures for presentation within your section.**

fitting template script is already configured to generate graphs of your data, the fitted curve and the residuals. However, as previously mentioned you will need to customize the titles and axis labels so they are relavent and meaningful to your graph. In addition, you may also wish to annotate your graph with additional information. **The fittemplate.m script has several examples of how to automatically place properly formatted fit results within a plot.** You can do this graphically with the tools provided in the graph window or by using the `'text'` command within the script. To use the `'text'` command you simply specify x and y coordinates and the string you wish to appear. The coordinates should be in the same units of the graph to which they refer. For example:

```
text(18,5,'y(x) = ae^{-bx}+a_1e^{-((x-b_1)/
    c_1)^2}+a_2e^{-((x-b_2)/c_2)^2})
```

would provide an appropriate label for the gauss3 graph in the lower left-hand corner. Notice also that Matlab can interpret latex formating to display Greek characters, superscripts, and subscripts.

When you have your graphs just the way you (and your section instructor!) want them (a completed graph might look like Figure 1) , you can output them for use in your lab notebooks, written summaries, and oral presentations. If you simply want printouts of your graphs for your notebooks, select `'Print...'` from the `'File'` menu and make sure the correct printer is specified.

If you wish to output your graph as a postscript file for use in a written summary or oral presentation, select `'Export...'` from the `'File'` menu and enter the file name you wish to use. You have a choice between many different file types for the export, but EPS (or EPS color if appropriate) will be the most convenient choice if you plan to use the graph in a latex document.

6. GETTING STARTED WITH MATLAB AT MIT

Throughout the year, and especially at the beginning of the academic year, Athena Minicourses on Matlab are offered, see `http://web.mit.edu/minidev/www/`. If you can't make one of these (or in addition), a very nice introduction to using Matlab on Athena is available at `web.mit.edu/olh/Matlab/Matlab.html` and has links to more extensive resources at MIT and elsewhere. Finally, talk to friends and classmates; many of them have a great deal of Matlab experience.

[1] Bevington, P.R., and D.K. Robinson, *Data Reduction and Error Analysis for the Physical Sciences, 3rd Ed.*,WCB/McGraw-Hill, Boston, 2003

[2] Levenberg, K., "A Method for the Solution of Certain Problems in Least Squares", *Quart. Appl. Math*, Vol. 2, pp. 164-168, 1944

[3] Marquardt, D., "An algorithm for Least Squares Estimation of Nonlinear Parameters", *SIAM J. Appl. Math*, Vol. 11, pp. 431-441, 1963

[4] Branch, M.A., T.F. Coleman, and Y. Li, "A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems", *SIAM Journal on Scientific Computing*, Vol. 21, Number 1, pp. 1-23, 1999

APPENDIX A: OTHER MATLAB USES IN JUNIOR LAB

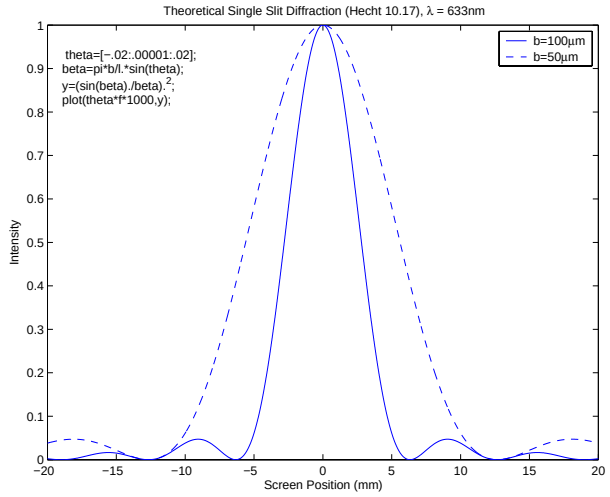

FIG. 2: Whenever possible, try to simulate the expected data by plotting some functional form which incorporates the essential physics of the problem. Here is a single slit diffraction experiment simulation.

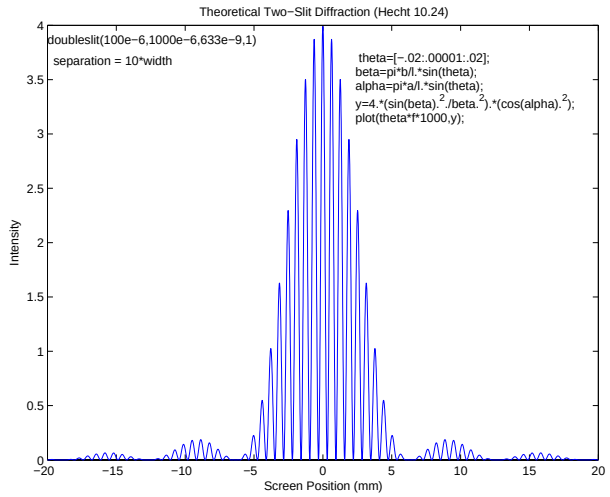

FIG. 3: And now, a double slit simulation...

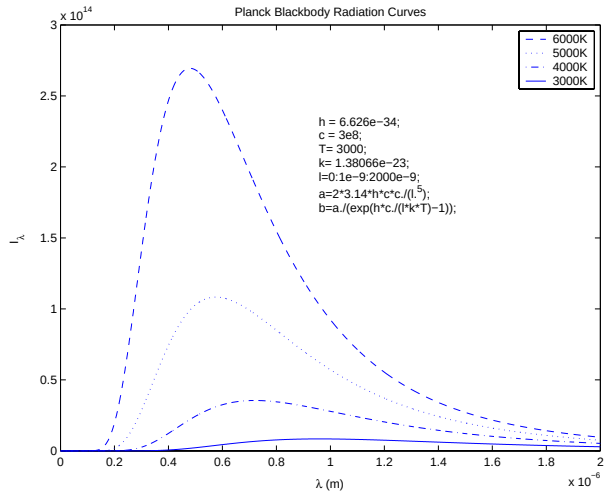

FIG. 4: Planck Radiation Curves

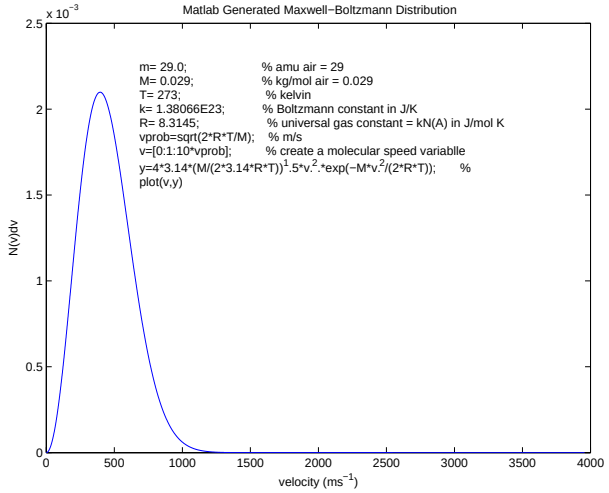

FIG. 5: A Matlab generated Maxwell-Boltzmann distribution

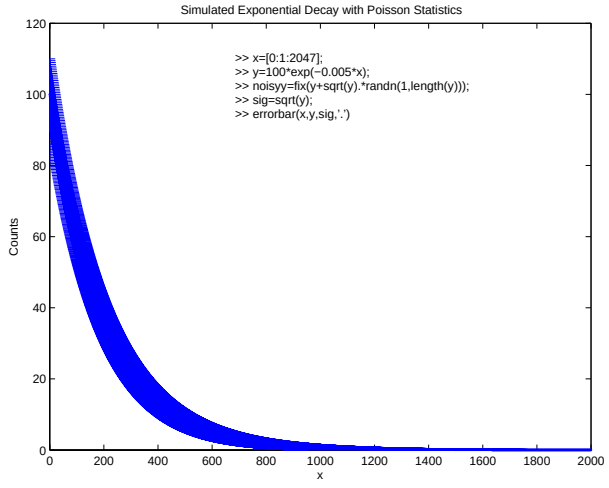

FIG. 6: A Matlag generated dataset simulating an exponential decay where the uncertainties are described by Poisson statistics.