**ENGR 695 Advanced Topics in Engineering Mathematics Fall 2023**

# Lab #2: Curve Fitting in *Matlab*

## Introduction

The goals of this lab exercise are to:

1. Practice developing a least-squares solution using the normal equation and compare the results to those obtained using the *Matlab* backslash (\) operator.
2. Introduce the *Matlab* curve-fitting command `polyfit` and explore its utility and some of its disadvantages.
3. Introduce the concept of the matrix condition number.

Before beginning, download the *Matlab* script `Lab2start.m`, which is available at the course Moodle site in the "Lab Materials" section. You should set up a separate folder on your own computer and/or in your Bucknell private Netspace for your ENGR 695 lab activities.

You might also want to locate and keep handy the last page of the Lab #1 handout entitled "Important Matlab Commands for Linear Algebra." It should be a very helpful resource.

## Background

An overdetermined system of linear equations is almost always inconsistent and therefore cannot be solved in the sense that an $N \times 1$ solution vector cannot be found that satisfies the $M$ equations in $N$ unknowns, where $M > N$. However, a "closest" solution usually can be found in which the solution vector comes closer to satisfying the $M$ equations than any other solution vector. As we have seen in class, the term "closest" has to be defined to be meaningful. In the case of the least-squares solution approach, "closest" means that the element-by-element differences (squared) between the actual values of the data vector $y_i$ (the right-hand side of the system of equations) and the estimates $\hat{y}_i$ of the data are minimized. The minimization process leads to a matrix expression for the closest solution that is often referred to as the *normal equation(s)*. Both the singular and plural forms of the term appear in the math literature.

An important application is the task of fitting known functions to a set of data. Often the data are noisy and could have unexpected trends. Much insight can be gained if a curve fit can be accomplished using simple functions. If so, the results might suggest a path for understanding the physical processes that produced the data. In other cases, it might be necessary to represent the data compactly via a weighted sum of a small number of elementary functions. There are numerous other applications as well.

In a typical problem, a weighted sum of functions is sought that best models the data. The functions are known and are typically elementary, and appropriate weights must be found to obtain the best representation of the data set. The problem can be cast in functional form as

$$y(x) \approx \hat{y}(x) = \sum_{j=1}^{N} c_j f_j(x),$$

where $y(x)$ represents the true behavior of the dependent variable, $\hat{y}(x)$ is the best ("closest," in a least-squares sense) fit to $y(x)$, $\{f_j(x)\}_{j=1 \text{ to } N}$ is a set of $N$ elementary functions, and $\{c_j\}_{j=1 \text{ to } N}$ is the set of coefficients to be found. In some cases, the function $y(x)$ is known but complicated, and the goal is to represent it using a sum of simpler functions. An example is a Fourier series representation of a complex periodic function. A Fourier series is a weighted sum of sine and cosine functions. To form the matrix expression that is required to find the coefficients, the function $y(x)$ must be sampled; that is, $y(x)$ is evaluated at $M$ known values of $x$, where $M > N$. In other cases, the actual function $y(x)$ is not known, but a set of $M$ discrete data points represents it.

In matrix form, the weighted sum above can be expressed as

$$\hat{\mathbf{y}} = F\mathbf{c},$$

where element the $ij$ of the matrix $F$ is given by $F_{ij} = f_j(x_i)$ for $i = 1$ to $M$. That is, the $j$th column of $F$ contains the $M$ samples of the $j$th basis function $f_j(x)$. Thus, $F$ is $M \times N$ in size. The $M \times 1$ vector $\hat{\mathbf{y}}$ represents the best estimates of $y(x_i)$ at the $M$ data points. The actual values of $y(x_i)$ are stored in the $M \times 1$ vector $\mathbf{y}$. Note that the values of $\mathbf{y}$ are initially known but the values of $\hat{\mathbf{y}}$ are not. If desired, the latter vector can be calculated after the $N \times 1$ coefficient vector $\mathbf{c}$ has been obtained to compare it to the actual data vector $\mathbf{y}$. Typically, the values contained in $\mathbf{y}$ and $\hat{\mathbf{y}}$ are plotted on the same graph to obtain a visual depiction of the data fit.

In the least-squares approach, a residual or error vector is defined as $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$, where each element $r_i$ is the difference between the actual data value $y_i$ and the best fit value $\hat{y}_i$. The vector $\mathbf{r}$ is $M \times 1$ in size. The goal of the least-squares method is either to minimize $|\mathbf{r}|^2 = \mathbf{r}^T\mathbf{r}$ or to make the residual vector orthogonal to the approximation; that is, to ensure that $\mathbf{r}^T\hat{\mathbf{y}} = 0$. Either approach leads to the *normal equation* given by

$$F^T F\mathbf{c} = F^T\mathbf{y} \quad \rightarrow \quad \mathbf{c} = \left(F^T F\right)^{-1} F^T\mathbf{y},$$

which uniquely determines a set of coefficients (contained in the $N \times 1$ vector $\mathbf{c}$) for the weighted sum approximation. Once the coefficients have been obtained, a plot of the best fit curve can be obtained by performing the matrix product

$$\hat{\mathbf{y}} = F\mathbf{c}.$$

Procedure

Start *Matlab*, and change the current folder to the one in which you saved the file `Lab2start.m`. Then open `Lab2start.m` in the *Matlab* script editor using the "Open" menu item in the ribbon at the top of the main *Matlab* window. You will be able to view and edit the file there.

The first several lines of the *Matlab* script `Lab2start.m` define a set of 21 data points that relates the dependent variable $y$ to the independent variable $x$. Your goal will be to fit to the data the function

$$\hat{y}(x) = c_1 + c_2 \sin(2x) + c_3\sqrt{x}.$$

The next few lines of the script are comments that indicate the places (marked by triple asterisks ***) where you are to insert new lines of code to find the required coefficients by:

1. using the *Matlab* backslash (\) operator,
2. using the normal equation as described in the "Background" section above, and
3. using the *Matlab* `polyfit` command.

There will be a lot of common code between the two sections, but the three solution methods should be distinct in the modified script that you produce. Note that the most difficult part of the first two methods is to form the matrix *F*. The same matrix can be used for both methods.

The third section of the script marked by asterisks requires you to use *Matlab's* `polyfit` command to generate a polynomial fit to the data. You may specify the order of the polynomial by changing the value assigned to the variable `Norder`. The `polyfit` command takes the *x* and *y* data that you supply and generates a polynomial of the specified order that best fits the data. In principle, the order can range from 1 (linear fit, in which the polynomial is $y = c_0 + c_1 x$) to an arbitrarily high number; however, as we will see, the order is limited by practical considerations. Note that the number of coefficients is one greater than the order.

Internally, the `polyfit` command applies the backslash operator to solve the overdetermined system of equations that defines the unique set of coefficients. If you examine the subsequent code in the *Matlab* script, it will help you to understand why polynomial fits can pose problems as the order of the polynomial becomes large. The script calculates the $F^T F$ matrix (which is $N \times N$ in size) that would be used in a solution based on the normal equation. The script then calculates the *condition number* of the matrix, which, as explained in the *Matlab* help facility, is "the ratio of the largest singular value of [the matrix] to the smallest. Large condition numbers indicate a nearly singular matrix." In other words, a large condition number indicates that any calculations that use the matrix are prone to errors, partly because of the wide range in the orders of magnitude of the matrix elements and partly because of the finite precision of the computer used to make the calculations. The lowest possible condition number is one, which corresponds to the identity matrix or scaled identity matrix (the most nonsingular matrix possible), but it can have values that reach double and triple-digit powers of 10 (approaching singularity).

After you have successfully computed the coefficients in the function above using the backslash and normal equation methods, complete the following steps using the `polyfit` command:

1. Run your script with the polynomial order set to the values 1, 3, 5, 10, and 20 (i.e., five separate runs).
2. For each case, save the resulting *Matlab* figure in a format that you can import into your favorite word-processing software (e.g., bmp, png or tif). The figure includes the solutions obtained using the backslash operator and least squares in addition to the `polyfit` function. The first two are the same for each case since their parameters don't change. Only the order of the `polyfit` solution changes from one figure to the next.
3. Note the condition number obtained in each case and whether it rises, falls, or randomly varies with the order of the polynomial. Briefly comment on your observations.
4. Run the script one more time with the polynomial order set to the value 21 (a polynomial with 22 terms, which is one more than the number of data points). Note the message that appears in the *Matlab* command window. Do not save the figure if one appears.

Assistance with each solution approach will be provided as needed, but try to deduce on your own how to complete as much of the work as possible.

After you have completed the lab activities, e-mail to me your modified *Matlab* script (m-file) with the file name `LName_Lab2_fa23.m`, where `LName` is your last name (surname), and the document containing the saved figures, your comments on how the condition number varies with polynomial order, and any other observations that you wish to share. Use the same naming convention for the document (e.g., `LName_Lab2_fa23.docx`).

<u>Lab Scoring</u>

Your score will be based primarily on the *Matlab* script and the document with figures that you submit according to the rubric posted on the Laboratory page at the course web site.

If you do not complete the exercises during the lab session, you may submit your documentation as late as 5:00 pm on Friday, September 8. If the files are submitted after the deadline, a 5% score deduction will be applied for every 24 hours or portion thereof that the item is late (not including weekend days) unless extenuating circumstances apply. No credit will be given five or more days after the deadline.