ENGR 4/695 Advanced Topics in Engineering Mathematics Fall 2024

Lab #1: Curve Fitting Using the Normal Equation

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 `Lab1start_fa26.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 495/695 lab activities.

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

Background

An overdetermined system of linear equations is one in which the number of equations M exceeds the number of unknowns N; that is, $M > N$. Overdetermined systems are almost always inconsistent and therefore cannot be solved in the sense that an $N \times 1$ solution vector cannot be found that satisfies all of the M equations. 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, which 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. In functional form, the problem can be expressed 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 (weights) 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 ij of the matrix F is given by $F_{ij} = f_j(x_i)$ for $i = 1$ to M. That is, the jth column of F contains the M samples of the jth 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 vector $\mathbf{c}$) for the approximation. Once the coefficients have been obtained, a plot of the best fit curve can be obtained by evaluating 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 `Lab1start_fa26.m`. Then open `Lab1start_fa26.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. If you wish, you may "undock" the editor window via its menu marked by three vertical dots.

The first several lines of the *Matlab* script `Lab1start_fa26.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 normal equation as described in the "Background" section above, and
2. using the *Matlab* `polyfit` command.

Depending on your level of familiarity with *Matlab*, forming the matrix F for the least squares (LS) solution using the normal equation could be a little challenging, so here is a hint:

The x data is contained in an $M \times 1$ vector x. The Matlab command `sin(2x)` produces an $M \times 1$ vector containing the results of the operation $\sin(2x)$ applied element-by-element to the data in `x`. Also, vectors can be placed side-by-side to form matrices by listing them within square brackets with spaces between the vector names. For example, if `a`, `b`, and `c` are 7×1 vectors, the *Matlab* operation `[a b c]` forms a 7×3 matrix. If you need a column filled with identical constants, see the help information on the `ones` command by typing `help ones`.

The second 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 uses the x and y data that you supply to generate 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 $y = c_1 + c_2 x$) 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.

The `polyfit` command applies the *Matlab* backslash operator to solve the overdetermined system of equations that defines the unique set of coefficients. If you examine the subsequent code in the `Lab2start_fa26` 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^TF matrix (which is $N \times N$ in size) that would be used in a solution based on the normal equation. For example, if the polynomial order were 15, then the F matrix would be 21×16 in size, and F^TF would be 16×16. The first column of F would contain the value 1 everywhere (x raised to the 0^{th} power), but the 16^{th} column would contain each of the 21 values of x raised to the 15^{th} power. The elements of F would span an enormous range of orders of magnitude, from the value 1 to the value 10^{15}.

The script then calculates the *condition number* of the F^TF matrix, which, as explained in the *Matlab* help documentation, 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 limited precision (i.e., finite number of digits) of values and calculation results that are stored in the computer's memory. The problem is that the F^TF matrix has rows that are almost linearly dependent; if they were plotted as vectors in N-dimensional space, the vectors would point almost in the same direction. 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). By comparison, the condition number of the F^TF matrix generated for the normal equation solution in this case is approximately 51.

After you have successfully computed the coefficients for the three-term function using the normal equation, 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 normal equation (least squares) and using the `polyfit` function. Only the order of the `polyfit` solution (and therefore the shape of the corresponding curve) changes from one figure to the next.
3. In a brief commentary section at the end of the document in which you have saved the plots, note the condition number for the `polyfit` solution 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, and explain in your document what you think the message implies. 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_Lab1_fa26.m`, where `LName` is your last name (surname), and the word-processed document converted to PDF format that contains 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_Lab1_fa26.pdf`).

Lab Scoring

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 11:59 pm on Monday, September 21. If the files are submitted after the deadline, a 10% 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 four or more days after the deadline.

Important Matlab Commands for Linear Algebra

The table below summarizes some of the more common operations that are specific to the field of linear algebra and their *Matlab* equivalents. Unless otherwise specified, vectors are assumed to be column vectors and contain N elements while matrices are square and $N \times N$ in size.

Action	**Example**	**Command(s)**	**Comments**
Define a matrix	$A = \begin{bmatrix} 1 & 5 \\ 9 & 4 \end{bmatrix}$	`A = [1 5 ; 9 4]`	; forces a new row
Define a column vector	$\mathbf{b} = \begin{bmatrix} 5 \\ 9 \end{bmatrix}$	`b = [5 ; 9]`	`'` is the transpose operator
		`b = [5 9]'`	`.'` transposes w/o conjugation
Get matrix size	M = rows(A) N = cols(A)	`[M,N] = size(A)`	see also `length`
$N \times N$ identity	I_N	`I = eye(N)`	`eye(M,N)` also works
Vector inner product	$s = \mathbf{x}^T\mathbf{y}$	`s = x'*y`	scalar result
Vector outer product	$A = \mathbf{x}\mathbf{y}^T$	`A = x*y'`	matrix result
Matrix transpose	A^T or A^H	`A'`	includes conjugation if complex
Matrix power	A^n	`A^n`	must be square
Matrix multiply	$C = AB$	`C = A*B`	inner dimensions must match
Hadamard (array) product	$C = A \circ B$	`C = A.*B`	matrices same size `./` and `.^` are similar
Matrix inverse	$B = A^{-1}$	`B = inv(A)` `B = A\eye(size(A))`	use only for "small" problems recommended if inverse needed
Solve $A\mathbf{x} = \mathbf{b}$	$\mathbf{x} = A^{-1}\mathbf{b}$	`x = inv(A)*b` `x = A\b`	not recommended generally recommended ("A under $\mathbf{b}$")
Determinant	$\lvert A \rvert$	`det(A)`	mostly for "small" problems
Matrix rank	rank(A)	`rank(A)`	uses the SVD (see below)
LU decomposition	$A = LU$	`[L,U] = lu(A)`	see `help lu`
QR decomposition	$A = QR$	`[Q,R] = qr(A)`	see `help qr`
Singular value decomposition	$A = U\Sigma V^T$	`[U,S,V] = svd(A)`	see `help svd`
Compute eigensystem	solve $A\mathbf{x} = \lambda\mathbf{x}$	`[V,D] = eig(A)`	see `help eig`
Extract column k from A	$\mathbf{c} = \mathbf{a}_k$	`c = A(:,k)`	column vector
Extract row j from A	$\mathbf{r}^T = \mathbf{a}_j^T$	`r = A(j,:)`	row vector