EE 105: MATLAB as an Engineer's Problem Solving Tool

Jay Farrell, College of Engineering, University of California, Riverside

December 20, 2023

Abstract

MATLAB is a valuable engineering tool that is used in almost all UCR EE courses. Proficiency in Matlab will provide long term benefits through out your career.

The objective of this laboratory is to familiarize the student with the usage of MATLAB. Most importantly, the goal is to show students how to use the Matlab *help facilities* to teach themselves how to learn about Matlab.

Between online search (e.g., google) and Matlab's help you can teach yourself anything that you need to do. For example, if you need to integrate a function, google 'Matlab integrate'. The search will return with links to both numeric and symbolic integration methods, including the Matlab function names, and examples.

1 Background

Matlab is an engineering tool used for computations, including matrix algebra and simulation.

This lab will show you how to figure out command names for functions and will familiarize you with Matlab through a few simple programming exercises.

2 Matlab Tutorial

UCR has a campus license to Matlab for educational purposes. Instructions for installation are on the BCOE systems website. Install Matlab on your computer well before your first lab session begins. Matlab is platform independent meaning that the same program (i.e., m-file) runs under Windows, MaxOS, or Unix.

1. **Start Matlab.** It may be slow to start as it has to communicate with a license server, but once it is started it should run very fast. If you get a message stating that no licenses are available, alert the TA who will contact the Systems Administrator.

2. **Find the Matlab help facility.** In Matlab, run "doc" in Command Window or click the "help" question mark on the home tab (F1). You should see a *Help Browser*.

   (a) The "MATLAB/Getting Started.../" is a great place to begin. Work through each subitem, making sure that you understand the portions of the desktop, entering matrices, matrix operations, matrix indexing, functions and m-files, plotting, working with complex numbers.

   - In the Matrices and Arrays section, make sure that you learn how to enter, add, transpose, and multiply matrices and vectors.
   - In the Programming and Scripts section, work through the 'Scripts' and 'Functions' parts to figure out: how to create/edit/run scripts, and how to define/edit/call functions.

   (b) The left side should show a table of contents. Getting comfortable with its use will help you throughout your career. Find the section 'Matlab/Graphics'. Teach yourself how to plot several curves on one plot with distinct line types. Add axis labels. Add grids and legends.[1]

   (c) Note that you can search for help in the 'search help' menu by entering the first few letters of the topic.

   Do not read passively. Instead, while reading, enter the examples and make them work[2].

3. **Matrices and Arrays.**

   Complete the following to make sure that you know the basic methods for using Matlab as a matrix manipulation tool.

   In the Matlab command window, use matrix operations to find the matrix C:

$$\begin{aligned} A &= [sqrt(2); exp(1; \pi)] \quad (1)\\ B &= [3; 5; 7] \quad (2)\\ C &= A^\top\, B \quad (3)\end{aligned}$$

   where a superscript $\top$ denotes a matrix transpose.

   ___Show your results to TA to record approval. ___

[1] In this course, and in your career, every figure in any report should: (1) have labels on the horizontal and vertical axes that give the name, symbol, and units of the variable plotted along that axis; (2) a descriptive caption; (3) a label such as "Figure 3" with a caption; and, (4) a reference by label in the body of the report such as "Figure 3 shows ...."

[2] You could also work through the help documents online even without MATLAB installed. Click here, or search "getting started with matlab".

4. **Command line help.** In addition to the help menu GUI you can get help on specific functions by typing 'help fname' where fname is the function of interest. Type 'help for' to get information about how to write a 'for loop.'

   A main challenge in using 'help' is that you need to know the correct name of the function. Online search is a great method to find the function name.

5. **Scripts.** When computing something that is more than one or two lines, the command window is not the recommended approach. If there is any error in the early commands, you would need to reenter the entire command sequence.

   Instead, work in a script (an m-file, not an mlx file). Scripts contain a list of Matlab commands (written by you or someone else). By writing a script, the process can be debugged, run repeatedly, and saved for future use. Because script name ends with '.m' they are referred to as 'm-files'. When you want to perform a sequence of Matlab commands, it is almost always preferable to use an m-file than to type the commands directly into the command window.

   Write a script that

   (a) clears the memory ('help clear');
   (b) defines the matrices A and B given above;
   (c) implements a 'for loop' to compute

   $$D = \sum_{i=1}^{3} a_i b_i. \tag{4}$$

   Your answer should be identical to C in the previous step, since the summation used in computing D is the definition of vector multiplication.

   Note: Matlab is designed to perform vector and matrix multiplications directly and efficiently. Whenever possible, use matrix vector operations (see eqn. (3)) instead of scalar operations with for loops (see eqn. (4)). In fact, whenever you are about to write a 'for loop' you should consider whether it can be written in matrix notation.

   __Show your results to TA to record approval. __

6. **More advanced Scripts.** From the Matlab window, you can open the Matlab editor by clicking on the 'new script' button in the upper left corner. M-files contain ascii text, so you can edit them with any text editor. Using the Matlab editor gives you some useful debugging and formatting features.

   (a) Create two m-files. Note that for Matlab to find your functions, you must use 'cd' in the command window to change the present working directory ('pwd') to the directory in which you have saved the function.

      i. The first m-file will have an input column vector x and an output column vector y where the i-th element of the output vector is $y_i = f(x_i)$ and

      $$f(x_i) = \frac{\cos(x_i)}{1 + \exp(3\,x_i)}.$$

      Your m-file should use vector math, not scalar processing (i.e., no "for" loops) and it must work for any dimension of the vector x. Use the help feature to learn about: cos, exp, "./", and ".*".

      ii. The second m-file will have an integer input N and no outputs. The m-file should define the vector x so that it contains $(N+1)$ equally spaced points on the interval $[0, 5]$, call the previous m-file to calculate the vector $y = f(x)$, and plot y as a function of x. The spacing between the elements of the vector x is $dx = \frac{5}{N}$.
      Start with a large value of (e.g., $N = 200$), run the function. Repeat this process with smaller values of N such as 5 or 10. Note that Matlab is not really plotting the function. It is just drawing lines between the (x_i, y_i) points. For the figure to look accurate, the value of dx must be small relative to the curvature of the function.

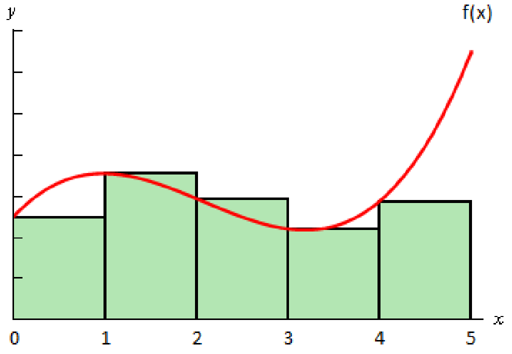
   Note: Any code that you turn in must be well-commented and well-organized or it will not be graded. *The code, figures, and discussion must be your own.*

   __Show your results to TA to record approval. __

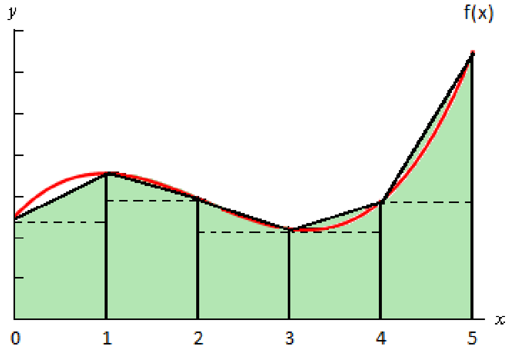
7. In this step, we will compare two approaches to numerically integrating the area under the graph of the curve $y = f(x)$. You already have a m-file to compute f from the previous step.

   See Fig. 1. In the following, I will assume that the name of that m-file is myfun.m.

   (a) Use the Matlab help feature to learn about integration routines and in particular the 'integral' function (or in previous Matlab versions use 'quad'). Use this function to approximate $Q = \int_0^5 f(x)dx$. Based on the help information, you should know the accuracy of this value of Q. In the following steps, you will form various estimates of Q and compare them with this value.



(a) Left rectangular approximation: Q_1.



(b) Trapazoidal approximation: Q_3.

Figure 1: Riemann integral approximation equations.

(b) Based on your calculus experience, you know that there are various methods to estimate the value of an integral:

$$Q_1 = \sum_{i=1}^{N-1} f(x_i)dx_i \qquad Q_2 = \sum_{i=2}^{N} f(x_i)dx_{i-1}$$

and

$$Q_3 = f(x_1)\frac{dx_1}{2} + \sum_{i=2}^{N-1} f(x_i)dx_i + f(x_N)\frac{dx_{N-1}}{2}$$

where $dx_i = x_{i+1} - x_i$ for $i = 1, \ldots, N-1$ and $0 = x_1 < x_2 < \ldots < x_{N-1} < x_N = 5$.

i. Similar to Figs. 1a and 1b draw a picture (try to draw it in Matlab using the 'bar' function) and use that image to explain the equation for the computation of Q_2. Label the figure appropriately and explain why the formula is valid only for small dx_i.

ii. What are the tradeoffs[3] related to decreasing dx? Think about the accuracy of the approximation and the amount of computation.

iii. Fig. 1b illustrates that Q_3 uses a trapezoidal approximation on each interval. For example, the area under the first line segment is $f(x_0)\,dx_0 + 0.5\,(f(x_1) - f(x_0))\,dx_0$.

A. Derive the above equation for Q_3.

B. Show that for equally spaced x_i (i.e., $dx_i = dx_{i-1}$): $Q_3 = (Q_1 + Q_2)/2$.

(c) The summation in the calculation of Q_1 can be written as the product of two vectors and calculated using vector arithmetic:

$$\begin{aligned} dx &= \frac{5-0}{N-1}; \\ x &= 0 : dx : 5; \\ y &= myfun(x); \\ A &= \left[\begin{array}{c} ones(N-1,1) \\ 0 \end{array} \right]; \\ Q_1 &= y \, * \, A \, * \, dx; \end{aligned}$$

where the semicolon is placed at the end in Matlab to prevent the result from printing to the screen (i.e., the code will be faster since screen printing is slow). The above should be working Matlab code.

Note that the value of Q_1 is a function of N. Implement the above code. Then implement another function which computes and plots Q_1 for all integer values of N from 2 to 200. Plot $Q_1(N)$ versus N. Also plot the constant value of Q determined by the Matlab built-in 'integral' or 'quad' function. Compare the two results. Also plot the error between $Q_1(N)$ and the constant value of Q determined above. Use 'subplot', to show these two graphs in the same figure window.

(d) Write your own code similar to the above to compute Q_2 directly; clearly state your definition of the matrix A. Compute Q_2 for all integer values of N from 2 to 200. Plot $Q_2(N)$ versus N on the same graph as Q_1 versus N. Also plot the error in Q_2 versus N on the same graph as the error in Q_1.

If your implementation is correct, they should converge toward the same value?

___Show your results to TA to record approval. ___

[3] In engineering decisions (and in life) it is rarely the case that a decision only has benefits or only has drawbacks. Instead, it has both. Before making an engineering decision, the designer should make a list and consider the tradeoffs between benefits and drawbacks.

8. **Read the article** “MatlabSpeedUp.pdf” that is attached. Implement and compare the various code segments. It is useful to understand and utilize these good programming practices.

9. **Run “demo”.** This will open a new window that allows you to select among various demos. Several of these may be useful over the duration of this course and the remainder of your time at UCR. Matlab will be used in the majority of your future courses, so learning it well now will help you in the future. Note that the ‘Simulink’ demo will be useful in the later part of this quarter.

3 Lab Report

Lab exercises should be done by each individual on their own computer. Discussion is okay. Copying (e.g., code, figures, or text) is a violation of student conduct.

This week’s lab report should only be a few pages long. Answer the questions from the lab. Include and discuss the graphs that are requested. There is very little code that actually needs to be turned in.

3.1 Grading of this lab report

The TA will describe the method that will be used to grade the report. Points will be given for clarity and correctness of the figures, code, and report discussion.

4 Lab Report Guidelines

The following guidelines apply to every lab report that you will turn in this quarter. The guidelines will not be repeated in each lab description.

1. Each student (not each group) must turn in his or her own lab report.

2. Every lab report should be as short as possible while still communicating to the grader that you understand all aspects of the lab. Excess length or printouts will lower your grade.

3. Proper English grammar will increase your grade.

4. The report must include all theoretical derivations with each step of the derivation clearly explained.

5. All variable must be completely defined (i.e., meaning, units, symbol, sign convention).

5 Prelab Rules

Each week, upon entering the lab you must show your prelab to the TA. The TA will keep a record of who has been signed off. Anyone not signed of in the first 15 minutes of lab will not receive credit for the prelab.

This week, there is no prelab.

6 Figure Guidelines

The following guidelines apply to every figure that you use this quarter. The guidelines will not be repeated in each lab description.

1. See footnote 1. If any of its specifications are ignored, the grader may ignore the figure.

2. The extent of the axis should be optimized to communicate the maximum amount of information.

3. The type of graph (e.g., loglog, semilog, rectangular, hist) used should be carefully considered. For example, the following sequence of data:

t -	1	2	3	4	5	6	7
x -	1e5	3	.5	.05	.005	.0005	.000005

   conveys much more information using the ‘semilogy(t,x)’ graph than the ‘plot(t,x)’ graph. Try both and see for yourself.

4. Use ‘subplot’ to fit various related graphs on the same page to make comparison easy and to save trees.

7 Code Guidelines

Avoid componentwise operations on vectors and matrices to the greatest extent possible.

Any code printouts that you turn in,

1. must be well organized

2. must be well commented

3. must be discussed by name in the lab report.

Speeding Up Matlab Programs by Orders of Magnitude

MARK C. MOLARO and RICHARD D. BRAATZ

Some tricks and tips on the usage of Matlab were provided in the August 2013 issue of *IEEE Control Systems Magazine* [1]. This column gives an example in which application of two tips speeds up a Matlab program by orders of magnitude. The example can be used to teach control engineering students the importance of following the two tips but also, more generally, the importance of implementation details when solving problems with a computer.

Consider a Matlab program that assigns the number one to each element of a row vector x (the clear command is used in each program to ensure a consistent basis for comparison):

```
>> clear x
>> tic
>> for i = 1:1000000
>>     x(i) = 1;
>> end
>> toc
```

It is fun to ask students how long they would expect this Matlab program to run. The tic-toc command reports the time taken to run this Matlab program, which was ~7000 s on a laptop with a Quad Core Intel i7 processor, with clock speed of 1.6 GHz with turbo boost up to 2.8 GHz, running Matlab Version 2010b on the Microsoft Windows operating system. The runtime was ~0.77 s on a similar computer running Matlab Version 2011a. The runtime for this simple assignment in 2010b was 2 h! At just under a second, 2011a is much faster but still very slow for the simplicity of the assignment. A good classroom assignment is to ask students why the Matlab program is so slow.

The primary reason the Matlab program is slow is because the dimension of the vector was not explicitly defined. Although common, this bad practice caused 2010b to redefine the dimensions of the vector x at each iteration of the loop. Version 2011a has a heuristic that reduces the number of redefinitions but still results in a long runtime. One valuable Matlab tip [1] is to define the dimensions of all arrays before they are used.

This column gives an example in which application of two tips speeds up a Matlab program by orders of magnitude.

The runtime of the Matlab program is greatly reduced by just inserting a single line that defines the dimensions of the vector x before being used:

```
>> clear x
>> x = zeros(1,1000000)
>> tic
>> for i = 1:1000000
>>     x(i) = 1;
>> end
>> toc
```

The tic-toc command reported a runtime of ~0.071 s for 2010b, which is about five orders of magnitude faster than the original program. The runtime for 2011a for this program was about ~0.030 s, which is more than one order of magnitude faster than the original program. Intuitively, both times are still very slow given the simplicity of the task. The problem is that the Matlab program uses a loop, and Matlab is slow at running loops. When Matlab encounters a loop, each statement in the loop is executed one after the other, without restructuring the computation to be faster. One tip is to avoid loops as much as possible [1]. Matlab has numerous built-in commands that allow the user to remove loops. One way of doing this is to use Matlab's indexing capabilities:

```
>> clear x
>> tic
>> x(1:1000000) = 1;
>> toc
```

The tic-toc command reported a runtime of ~0.014 s in 2010a, which means that a fivefold reduction in runtime occurred

Digital Object Identifier 10.1109/MCS.2013.2279486
Date of publication: 14 November 2013

(continued on p. 163)

2016

▲ **AMERICAN CONTROL CONFERENCE (ACC)**
6–8 July
Boston, Massachusetts, USA
General Chair: Danny Abramovitch
Program Chair: George Chiu

▲ **IEEE MULTICONFERENCE ON SYSTEMS AND CONTROL (MSC)**
TBD
Buenos Aires, Argentina
General Chairs: Ricardo Sánchez-Peña and Mario Sznaier
Program Chair: TBD

▲ **IEEE CONFERENCE ON DECISION AND CONTROL (CDC)**
December
USA
General Chair: Alessandro Giua
Program Chair: Francesco Bullo

FOCUS ON EDUCATION

(continued from p. 135)

by removing the loop. Version 2011b gave a similar runtime of ~0.011 s. An even faster program in 2010b is obtained by using a Matlab's built-in command:

```
>> clear x
>> tic
>> x = ones(1,1000000);
>> toc
```

The tic-toc command reported a runtime of ~0.0067 s, which is more than ten times faster than the previous program. For 2010b, the final program is more than *1 million times faster* than the original program. The runtime for this program in 2011a was ~0.027 s, which indicates that its use of the ones command actually resulted in longer runtimes. For 2011a, the runtime of ~0.011 s for the fastest program was nearly two orders of magnitude faster than the ~0.77 s obtained by the slowest program.

The above results indicate that the runtime of a Matlab program can be very sensitive to version number and can result in a switch in ranking in the relative runtimes between two programs. Version 2011a resulted in lower runtimes when matrices are not preallocated, but this cost is still substantial.

Matlab's built-in commands are usually much faster than achieving the same result from scratch. The built-in commands cover a wide variety of operations and include finding maximum or minimum values in arrays, cumulative sums, cumulative products, and norms of vectors and matrices. Matlab is fastest when the programs are implemented as a combination of matrix-vector operations and built-in functions. When encountering a runtime in a Matlab program that is much longer than seems reasonable, one place to search for solutions is the Matlab blogs [2].

Matlab's built-in commands are usually much faster than achieving the same result from scratch.

REFERENCES

[1] K. K. Leang, "Matlab tricks and tips," *IEEE Control Syst.*, vol. 33, no. 4, pp. 39–40, Aug. 2013.
[2] Matlab central blogs. (2013). [Online]. Available: http://blogs.mathworks.com/