Kevin Chen

12/20/16

Matlab Notes

General Concepts

* Understanding the MATLAB interface
  + Folder directory on the left, command window in center (type clc to clear the window), top right is workspace tile (contains list of all active variables in your simulation. Change the information available about your variables by right-clicking by the columns and select the pieces of information you want to display. Clean up the workspace by clicking the drop down arrow on the top right of the panel and click clear workspace), bottom right is command history (can see commands you sent in past, and can copy-paste previous commands. Clear it with clear command history in drop down menu).
  + Top bar can create new script, new file, or open file. Environment tab can be used to change the relative position of the tiles in MATLAB and the default active folder. Resources tab can access online documentation.
* Working with MATLAB variables
  + Type in command variable.
  + Don’t add a variable declaration statement. (Don’t declare type.)
  + Type A = 1. This causes the command window to show the result that A = 1, command history has new command, and workspace has new variable.
  + Can choose to display variable class to see what type A is.
  + If you want to create a variable with a different class, manually input the class of the variable by including the class as a function in the variable definition. Example: B=uint16(1). See all class names though help class.
  + Repeat above step to a variable that was already created to change the variable class’s name. Some class transformations cause information to be lost.
  + Create a vector by surrounding numbers separated by spaces or commas with brackets. Example: C=[1 2 3] or C=[1,2,3]
  + Create a matrix by creating a vector with semicolons separating rows. Example: D=[1,2;3,4]. Can replace commas with spaces and semicolons with new lines.
* Everything is a matrix
  + A scalar is just a 1 x 1 matrix, and a vector is just a n x 1 or 1 x n matrix.
  + The exact same types of operations can be used on each of the different variables.
  + Anytime there is no ambiguity over what type of function you want, MATLAB interprets your function in the most appropriate way. Example: putting an asterisk between two matrices is interpreted as matrix multiplication.
* Understanding data structures
  + Creating larger dimension arrays: creates an n-dimensional array with dimensions and all entries containing zero. Can replace the word “zeros” with “ones” to fill them all with ones, “rand” to fill them with random numbers between zero and one, and randn to fill them normally distributed with all random numbers
  + MATLAB displays each 2D slice of the array. It is labeled as so: ans(:,:,dimension\_1, dimension\_2, …, dimension\_n) = 2D\_array\_representation
  + Can refer to each 2D slice using the same syntax. Example: a(:,:,1)=[1,2;3,4]
  + Can have a matrix containing matrices. Example: e={a,b;c,d}, where a, b, c, d, are submatrices of the “supermatrix” e.
  + Data structures: add fields to a data structure using the syntax: struct\_name.field\_name = value. Example: s.name = ‘Kevin’. This will create a 2D 1 x 1 structure named s with a field “name” inside the cell at location (1, 1).
  + Can expand dimensions of the structure by adding more elements. struct\_name(i,j).field\_name = value. Example: s(1, 2).name = ‘John’
  + Access elements as so: struct\_name(i,j). struct\_name would list all entries (if it is one dimensional).
  + Instead of specifying (i, j), you can just type in one number to refer to a cell. This is the number of each cell: . However, this is considered a bad idea if your matrix is 2 dimensional.
  + Can use colons to take an entire slice.
* Core MATLAB Syntax
  + Put single quotes around strings
  + nam2str(number) converts the number into a string. strcat(string1, string2) concatenates two strings.
  + To transpose a matrix, put an apostrophe after it.
  + Semicolon at the end of the line will suppress the output.
* Using built-in functions and variables
  + Online documentation: [www.mathworks.com/help/matlab](http://www.mathworks.com/help/matlab)
  + Type help function\_name if you need more info about a function.
  + Don’t use parenthesis when using a function without parameters. Only use them when the function does have parameters.
* Working with matrix and scalar operations
  + Let A and B be matrices, c be a scalar. A + B would add the matrices, A + c would add c to each entry of A (scalar addition).
  + A\*B would perform matrix multiplications, A\*2 would perform scalar multiplication.
  + Adding a dot in front of any operation (+,-,\*,/,^,=) tells MATLAB to execute the operation piecewise. That means perform this operation on each pair of corresponding cells between the two matrices.
  + Dot product: Transpose one of the matrices and do matrix multiplication
* Control Flow
  + if-statements:

if *condition1*

//Stuff to do if condition1 is true

else if *condition2*

//Stuff to do if condition2 is true

else

//Stuff to do otherwise

end

* + switch statements:

switch *expression*

case *value1*

//Code

case *value2*

//Code

otherwise

//Code

end

* + break and continue
* Understanding data types
  + Round-off errors: Can try to solve by:
  + 1. Change the data type to a more precise one.
  + 2. Symbolic arithmetic: MATLAB will consider data as rational fractions rather than decimals. This removes all rounding. Requires the use of the symbolic math toolbox. But this is significantly slower than floating pointing arithmetic. To use this, put sym and parenthesis around each of the numbers that you want to be a symbolic data type.
  + You must have data of the same type for MATLAB to do calculations with them.

Programming in MATLAB

* How are program files stored?
  + Current folder you are in is in top bar of the screen in a breadcrumb trail.
  + Use the top bar and the navigation menu to navigate to whichever folder you want
  + By default, MATLAB can only find .m files that are either included in the program by default or present in your currently active folder. That means if a certain file is referenced by your script, it must be in your currently active folder.
  + Files included by default in MATLAB are called the Search path. If you want MATLAB to look in additional folders, you can add them to the Search Path by typing addpath(*file\_path)* in the command window.
  + Since you don't always know where or when a script will run, it's a good practice to have your script call addpath with the path of any other function that it's planning on referencing.
* Viewing and editing programs
  + Click on “New Script” to create a new program. Run the program by clicking on the run button within the editing window – OR -- righting click on the script within the current folder window and choosing run or pressing F9 – OR – typing the script’s name in the command window and pressing enter.
  + The final above method is especially useful if your script is a function since this is the only way to provide your inputs.
  + When running a function, make sure you provide the correct inputs. MATLAB doesn’t have a variable definition statement, so it will not be able to tell you what format or dimensions you need for the variables. Instead it will attempt to run the function with whatever data is provided, which can lead to unpredictable behavior. Because of this, have descriptive variable names and add checks in your method.
* Creating scripts
  + MATLAB can automatically convert CSV files into matrices; you can simply use the csvread(*file*) function.
  + mean(*matrix*) takes the mean of each row in the matrix and uses those values to make a new matrix.
  + m:n creates an matrix with values m through n.
  + For loops example:

for i=0:9

//do something 10 times

end

* Creating functions
  + Click on New --> Function. Syntax is already created for you.
  + Make sure to assign output\_args to something if you want to return some values.
* Debugging
  + Can be done by removing semicolons.
  + Breakpoints can be added to the script editor by selecting the line you want a breakpoint on, clicking on the breakpoints drop-down menu, and then clicking on set/clear. Click on run and the program will stop on the lines with breakpoints. Click on step to go to the next line or continue to run until the next breakpoint.
  + Conditional breakpoints will only activate when a specific condition is met.
  + Checking the Stop on Error or Stop on Warning options causes the program to treat the error like it’s a breakpoint and just pause. (By default it will continue to execute if it runs into a warning or completely quits if it returns an error.) This can be helpful since the state of the program at the time when the error occurred is saved.
* Performance Considerations
  + Tic and Toc functions: tic() starts a built-in millisecond watch, and toc() returns the time elapsed since the last “tic”
  + Writing output to the window slows down the execution of programs substantially
  + Allocate a variable only once. It takes far less time for MATLAB to define a 100 x 100 matrix rather than defining a 1 x 1 matrix, extending it to 1 x 2, etc. Use the zeros(), ones(), randn(), or rand() functions to do this.
  + More precise data types take more time. Symbolic arithmetic is even worse.
* Adding program documentation
  + Add comments by adding a % sign. (Or use it to comment out code.)
  + Block comments: %{ at beginning, %} at end.
  + In-line comments: Commenting out a section of a line (with uncommented code on both sides of the commented section). This is done as so:

beginning\_of\_line…

…commented\_section…

end\_of\_line

* + Help comments: This is a comment that occurs at the very beginning of your script or right after the function definition of your function. It can be viewed without opening up the script or function itself by typing in help followed by the function name in the command window.

Data Representations

* Creating basic plots
  + plot() function plots a 2D graph. Various parameters to customize the plot.
  + Can pass in a vector and it plots the values against the indexes of the vector.
  + Can pass in two vectors to plot one on the x-axis and the other on the y-axis.
  + Once plotted, the graph can be edited in the tools dropdown menu and choosing edit plot – OR – running the Plot Edit command in the Command window.
  + After selecting edit plot, double-clicking any element of the plot will allow you to select and modify it.
  + MATLAB can plot three dimensional graphs by using the command plot3(variable1, variable2, variable3). The rotation tool may be useful to change the perspective on the graph.
* Adding annotations
  + xlabel(‘label\_name’), ylabel(‘label\_name’), zlabel(‘label\_name’) labels each of the axis.
  + title(‘title\_of\_graph’) adds a title to the graph.
  + Add a legend to a graph using the legend() function. Pass in a string for each data set separated by commas.
  + MATLAB considers the most recently plotted graph to be the active graph and all changes will affect it.
  + You can get the current figure handle with the gcf command by typing the line h=gcf with no parameters or turning an integer corresponding to the number of the figure displayed in the top bar of the figure window.
  + To select a point on a graph, use the Data Cursor tool and then click and drag to select a particular point.
  + You can save your graph by clicking file 🡪 save as
* Working with images
  + Import an image by double-clicking on the image in the current folder window.
  + Each image is a matrix with each entry representing a pixel. Most common format is RGB. As a result, MATLAB models this image as a three dimensional matrix, with the first two dimensions equal to the width and height and the third equal to three for each pixel to have the correct three RGB values.
  + Since MATLAB treats image data as nothing but a matrix of numbers, any formula that works with pixels of the available data type will function normally.
  + Some functions that work well with MATLAB images are: image(C) displays the data in matrix C as pixels in an image, imagesc(C) does the same as image(C) except that the colors in the image are scaled so that the highest and lowest values in the matrix cover full range of colors (basically increases the contrast), imread(file, format) loads an image from the file with the specified format as a matrix, imwrite(image, file, format) stores the MATLAB matrix as an image file of the specified format.
* Creating responsive programs
  + The pause function stops your script until the user presses any key. Passing in a number causes the program to wait that number of seconds before continuing.
  + Pausing can be disabled globally through the command pause(‘off’) command and then reenabled through the pause(‘on’) command.
  + Getting user input: use the function input(‘*message\_to\_display*’), and assign its value to a variable.
  + MATLAB automatically attempts to evaluate the entered text as a normal command, so users can reference variables, call functions, or perform arithmetic operations. If you don’t want the user to run commands, pass in an extra parameter, ‘s’, which will cause MATLAB to treat the input as a literal string.
* Editing variables manually
  + In the workspace, double click on any variable. This will bring up a table displaying the values in the matrix. Modify any value by double clicking on the appropriate cell. This doesn’t work if the dimensions of the matrix are too large or if the matrix is more than 2 dimensional.
  + For larger matrices, assign a portion of the matrix to a temporary variable. Modify the temporary variable in the workspace. Assign it back to the appropriate portion of the original matrix.
  + Can save the state of the variables without a script by clicking on “Save Workspace” in the home tab menu. Double clicking on that file will then recreate those variables with the exact same values.

External Toolboxes

* What are MATLAB toolboxes?
  + MATLAB has a wide variety of extensions and program libraries, called toolboxes, available for use in a diverse field of applications.
  + Some toolboxes such as the Symbolic Math Toolbox also includes separate apps and even separate programming languages.
  + Type ver into the command window to display which toolboxes are installed on your version of MATLAB
* Statistics Toolbox
  + All regressions techniques in this toolbox revolve around the LinearModel class. LinearModel.fit(indvar, depvar) displays a linear regression model with coefficients for the intercept and slope of the function. Also calculates standard errors, t-statistics, and p-values.
  + Regression techniques: linear, nonlinear, robust, ridge
  + Has functions for: analysis of variance (ANOVA), probability distributions, hypothesis testing. (Look at documentation for details.)
  + plot(model) graphs independent vs. dependent variable, finds the best-fit line, and generates confidence intervals.
* Symbolic Math Toolbox
  + Rather than calculating a single solution to an equation, the symbolic math toolbox allows you to input an equation and have MATLAB solve it analytically to provide the general solution. This also eliminates rounding errors and other side effects of finite computation power.
  + Designate a variable as symbolic by using the sym(*variable*) function. Represented as a rational function rather than a decimal.
  + Advantages: no round off errors and no infinite/repeating decimals. Disadvantages: slow to compute
  + To designate a function as symbolic use the symfun(*function*, [*inputs separated by spaces*]) function. This will define a general function that takes in the given inputs and combines according to the function to produce an output. Unlike normal functions, this can be used as an input term in another function. Example:

syms l w h % designates three variables to be used in the function

volume = symfun(l\*w\*h, [l w h]) % defines and stores the function

volume(l, w, h) % call the function anytime as so

* Optimization Toolbox
  + Aids in the estimation of variables and parameters in optimization problems (e.g. regressions).
  + optimtool: the Graphical Optimization Tool. Can optimize a wide variety of equations based on various criteria.
  + Access this application by typing in optimtool in the command window. Fill in the various fields and run the optimization. The optimization function is an @ symbol followed by the name of our script.
  + You can see how the optimization occurred by clicking Options 🡪 Plot Functions bar 🡪 check Function Value, and then run the function again.