Week 2 - Advanced Plotting

Reminder - the best place to learn MATLAB (or anything, really) is the internet! StackOverflow and MathWorks' own MATLAB Exchange are filled to the brim with people asking and answering questions about MATLAB. MATLAB's own documentation is also extensive and extremely helpful. It includes descriptions of how to call functions as well as usage examples.

Download this page as a PDF here (this document is generated automatically - print this page to PDF from your browser for the best result).

Return to Lectures

Plotting Functions

As an engineering student, it will be of great interest to you to be able to plot functions over a range to see how they behave. This could be anything from comparing the ideal gas law to the Peng-Robsinson equation of state or comparing concentrations across a gradient for a separation.

Let's start off with a simple function that returns twice the number we give it (aka *y=2x*): ```matlab function y = double(x): y = 2 * x; end

% note - for simple functions like this, you can use this syntax, similar to a Python lambda expression double = @(x) 2 * x % function_name = @(input_1, input_2, ...) operation; <- semicolon to supress output of assigning this function to "function_name" ``` *see MATLAB's documentation here for more on anonymous functions)*

You can then plot this function using a range of x values generated by using the *linspace* function (documentation here) or by using the below shortcut to the Python-like range function: ```matlab % syntax - start:step:stop (inclusive stop) x_range = 1:10 % becomes 1, 2, 3, ... 10 x_range = linspace(1,10,10) % generates the same list % linspace(start, stop, number_of_points) <- default number of points is 100 (a lot to us, not your computer!)

% evaluate the function over the interval y_values = double(x-values); % call the plot command plot(x_range, y_values) ``` Colon notation is helpful if you know exactly *which* values you want to see the function evaluated at. *linspace* is useful if you only care about overall trends/behavior or if you are only interested in the end behavior. Note that for logarithmic functions, the function *logspace* acts in the same way, except that it generates logarithmically spaced values.

One of the most common errors you will see when plotting functions revolves around Array vs. Matrix Operations and when you accidentally try to do one instead of the other. Remember that certain matrix based operations are undefined (such as division) and what you are probably after is *element-wise* operation, where you iterate through every element in the matrix and divide by a given scalar value. Element-wise operations are denoted with a **.**, such as ***./*** and ***.^***.

Customizing Plot Programmatically

Although MATLAB's property inspector is an excellent tool for fine tuning plots, reproducibility is limited and the slow. It is good practice to use the below commands to customize your plots in-line, rather than by hand after generating them. It is also a great idea to make a general plot customizing function that calls all related formatting commands at once, saving you time in the future - an example of this is included at the end.

title, xlabel, and ylabel

These three functions do just what one would expect. In a style similar to excel, they will take an input string and write it in the appropriate location on the plot. For advanced use of these functions, check the documentation. One good thing to note is that you can enter LaTeX-formatted strings as arguments to these functions and they will be interpreted into excellent looking math script, including super and sub-scripts.

xlim, xticks, and xticklabels

Calling these functions will enable you to edit the horizontal axis limits, tick values, and how said ticks are labeled. - *xlim* takes an linear array of two values, the lower and upper limit of the xaxis - *xticks* takes a linear array (one dimensional) of values for the x axis - *xticklabels* takes a cell array of strings, each of which is a label on the axis. Here's an example: `matlab plot(linspace(1,10,50),rand(1,10),'LineWidth',5) % make sure the vectors are the same size % it is easiest to set a thicker line when calling plot initially, as above xlim([3,7]) % maybe I decide to only look at one portion of the data xticks([3,5,7]) % only show odd points xlabels({"Day 1","Day 2"," Day 3"}) % can rename points to accurately reflect what they are`

ylim, yticks, and yticklabels

This group of functions is analogous the ones above, except that they operate on the vertical axis. I will point out that y tick labels are, by default, written horizontally for readability. If you want to rotate them, use *ytickangle* (and, for the horizontal axis *xtickangle*)

plot multiple lines on the same figure

There are two ways to plot multiple lines on the same plot. 1. When calling *plot*, provide the data as a 2D matrix. Each column will be plotted as a separate line. 2. After plotting your first line, issue the command *hold on*. Subsequent calls to plot will be placed on the same figure, until you type *hold off* or start a new figure (*figure()*)

gca and customizing axes, subplots

You can modify all aspects of the axes on a plot, from how long the ticks are to which direction they face. Use a command like the one below to access the *axes* object for the current plot. `matlab ax = gca;` From here, I recommend reading the documentation about *gca* to figure out what else you can modify.

Much like in matplotlib, you can also create multiple plots side-by-side in the same figure using subplot. I'll skip that here because it is generally easier to make multiple large plots and paste them next to eachother in whatever document you turn in, particularly for our engineering classes.

General Plot Customizing Functions

This code is shared as a *.m* file in the Code Examples repository with better in-line documentation link.

Notice that we do not have to pass in any reference to the figure to operate on when calling *makePretty*. This is because plotting functions in MATLAB will inherently call *gcf* (get current figure documentation which is not a variable but rather a function. Whenever you call it, it will retrieve whichever is the current figure. gcf returns the current figure handle. If a figure does not exist, then gcf creates a figure and returns its handle. You can use the figure handle to query and modify figure properties. `matlab function makePretty(intitle,inxlabel,inylabel) grid on box on ax = gca; ax.XAxis.LineWidth = 2; ax.YAxis.LineWidth = 2; title(intitle) xlabel(inxlabel) ylabel(inylabel) end`