Video: Getting Started with MATLAB

[00:01] CS332 uses vision software written in MATLAB, a powerful technical computing environment that's used in many disciplines, including those that we bring together in this course. In assignments, you'll write your own MATLAB code to analyze images, and aspects of the language that are most useful for this work are described in this online document that's linked from the course schedule page. This video introduces you to the MATLAB development environment.

So let's get started. I'm working on a Mac and have MATLAB in the toolbar at the bottom of my screen, but I can also go to the Applications folder in the Go menu and find MATLAB there. I have Release R2019b installed on my Mac. If you download Wellesley's current version, it might be a 2020 release - either of these versions are ok and you can install MATLAB on a Mac or PC.

[01:05] This is the MATLAB Desktop that you'll see when you first launch the application. There's a toolbar at the top with icons and dropdown menus, and subwindows that might include the Current Folder on the left, Command Window in the center, and Workspace on the right. The borders between the parts can be dragged around, like in any application, and the overall visual layout can be controlled with the Layout menu at the top. I'm just using the Default layout here. Each subwindow also has a dropdown menu at the far right end of its title bar that enables you to do things like close that window or undock it from the main Desktop.

[01:53] So what are these different components of the Desktop? MATLAB is interactive - you can type commands in the Command Window, for example, a simple assignment statement to create a variable (`num1 = 10`). MATLAB evaluates your commands and returns one or more values that may be printed out. There's a lot of vertical space in the printout here and we can avoid that by saying `format compact`. Then if we bring that variable back again, we see it's printed out more compactly. I'll create a second variable that's assigned to an arithmetic expression that uses the value of the first variable, maybe I'll square it (`num2 = 2 + num1^2`). Now you can guess what the Workspace is - it's the collection of current variables, which are listed in the Workspace window on the right. If you enter an expression but don't assign its value to a variable (`sqrt(num2)`), MATLAB assigns it to a generic variable named `ans` and also puts that in the Workspace. If you add a semicolon at the end of a statement (`ellen = 29;`) the variable is created, it's given a value, but the printout is suppressed. That semicolon is really important when we're working with large images. Let's load an image using the MATLAB function `imread` - (`image = imread('kittens.png')`) - but I'll forget the semicolon. So now you can see it has printed out all the values stored in the array that contains that particular image. If I bring back that command (I'm just pressing my up-arrow key to get my most recent command), but now I'm going to add that semicolon, and that's much better. We can enter `clc` if we want to clear out the printout that's displayed there, and all the variables are still in place and we can access them (`ellen`).

[04:18] What about the Current Folder on the left? At any one time, there's a Current Folder of files that we have direct access to. Below the toolbar, above the Command Window here, is the path to the Current Folder - here it's `Users>ellenhildreth>Documents>MATLAB` and there's nothing in that folder. Let's navigate to a folder with some content. I'll click on `ellenhildreth` in the path. I have these folders at the top level of my Mac and I'll double-click on `Desktop`, then on `MATLAB_intro`. [added note: There are also some icons above the Current Folder window that you can use to navigate around the folders on your computer. If you hover over them, there's a popup message about what they do.]

There's several files in this folder with different file extensions. Files with the `.m` extension are simple code files that we refer to as M-Files, and there are two types here. `myScript.m` is a script file. I'll double-click on that, which will bring it up in the editor, and the editor by default, will appear above your Command Window. This script contains a few simple commands and some comments in green that start with the `%` sign. We can, for example, create a one-dimensional vector or array of numbers by writing a set of values inside brackets, and then perform computations on these vectors, for example, sum up all their values, perform the same arithmetic calculation on each value in a vector all at once, or combine the contents from two vectors in an element-by-element way. When we run a script, the statements will be executed one-by-one as if they were entered directly into the Command Window. You can run a script in two ways. First, you can click on the green Run triangle at the top in the toolbar. All of the statements were executed and all the new variables were placed in the Workspace. The first two statements have semicolons, so their values didn't get printed out in the Command Window, but here they are (`vect1`, `vect2`). The other statements have no semicolon, so their values were printed out. We can also run a script by entering the name of the script in the Command Window. For example, I can write `myScript` without the `.m`, and it's executed again.

[06:58] There's another M-file here, `myFunc.m`, and that's a new function that I defined. Let's bring that into the editor. In the header of the function, you see its name `myFunc`, and note that the name of the function is the same as the name of the file. MATLAB assumes that functions that you define will have the same name as the name of the file - if it sees that name `myFunc` anywhere in the code, it goes hunting for a file with the same name to find its definition. The function has two inputs, `input1` and `input2`, and it also has two outputs called `arithMean` and `geomMean`, and it just performs calculations of the arithmetic mean and the geometric mean of two input numbers. [added note: For each output variable, we need to have at least one assignment statement in the body of the function that indicates the value to be returned through the output variable.] In this case, we can't actually run the function using the Run button at the top, because we need to supply values for those inputs (if I do, MATLAB tells me that, with an error at the bottom). I can run this function either in the Command Window or from some other code file, for example, `[mean1,mean2] = myFunc(5,10).`

[08:35] The file with the `.mlx` extension is a MATLAB Live Script, which is a lot like a Python notebook. Let's open that up. First, you'll see some new tabs at the top, with toolbars `Live Editor`, `Insert`, and `View`. A Live Script itself can include text, images, formulas, and you can also put in executable code boxes. We'll work with this particular Live Script in our first class, but just for a taste, let me place my cursor inside this first code box and click on the `Run Section` option in the toolbar, and the code is executed and the values are printed out right in place. We can go back and change statements here and re-execute that section. Any changes, any new variables that we create will appear in the Workspace with their new values.

Some of the demonstrations of visual processing algorithms that we'll run are interactive programs with a graphical user interface. They're called apps, and they have the `.mlapp` file extension. Let's run this simple app, and I can do that by typing the first file name in the Command Window (`imageApp`). Here's my graphical interface, and I can interact with that, I can make a selection of images, I can choose between kittens or an Ansel Adams photograph, let's load the kittens, and there it is displayed. There are boxes here that I can edit to change values, for example, let's suppose I make a neighborhood size of 8 and blur the imagem, and there's my blurry kittens. There's another demonstration at the bottom here, with a threshold. It's going to create a binary image where all the brightness values that are above 100 in this case will be displayed as white and anything below that will be displayed as black. So let's look at our binary image. When I'm done interacting with this program, I can click on the `quit` button. You can also see files here that are image files with extensions like `.png` or `.jpg`. With MATLAB you can work with many different image formats.

[11:23] A final thing I'd like to show you is how to abort a computation that's out of control. Let's go back to the original script file, and you can see there's tabs here that allow me to switch between any of the files that I've opened so far. If we go back to `myScript`, there's some code here that's initially commented out, but I will uncomment it by selecting it, and I'll use the `%x` icon at the top. You can see here that it creates a variable `i` that is assigned to 0, and as long as `i < 10`, it's going to subtract 1 - clearly not a winning strategy. Let's run that script again, and we see at the bottom of the Desktop, MATLAB is Busy - it's in this infinite loop. We can put it out of its misery by entering `control-c`.

So in this video, you learned about the logistics of starting MATLAB, the components of the Desktop, different kinds of code files, and just a tiny taste of the language itself. You can read much more about the MATLAB language in the document that I showed at the beginning, and you'll explore MATLAB in our Zoom classes, starting on the first day.